Generic framework for photoresist-based metrology calibrations

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

Generic Framework for photoresist-based metrology calibrations

Spyridon Strouzas
September 2013

TU Delft
Delft University of Technology

TU/e Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

Generic Framework for photoresist-based metrology calibrations

Redesign of calibration component and proof of the new architecture's feasibility

Spyridon Strouzas

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML

Eindhoven University of Technology

Steering Group

S. Strouzas
P.P.A.A. Peeters
J.J.C.P. Schuehmacher
J. Vazifehdan
H.T.G. Weffers

Date

September 2013

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 7.090, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

ISBN
978-90-444-1213-0

Abstract
ASML is a company that designs, develops and produces photolithography machines, called wafer scanners, used in the process of manufacturing chips and integrated circuits. In order to achieve this it requires nanometer accuracy at high speeds. For the nanometer accuracy to be reached, the system must have a highly accurate calibration system. The calibration is achieved both through hardware and software means. One part of this calibration process is the exposure of a test wafer, readout of the exposure results and adjustment of the machine's parameters. This sequence is executed repeatedly during the calibration phase. Test engineers need to be able to create new calibration tests for this sequence in a convenient way. This report describes the proposal for a new architecture of the exposure framework used by all the exposure calibration tests. Additionally, the process that drove this project is explained.

Keywords
ASML, calibration test, exposure framework

Preferred reference
Spyridon Strouzas, Generic Framework for photoresist-based metrology calibrations: Redesign of calibration component and proof of the new architecture's feasibility. Eindhoven University of Technology, SAI Technical Report, September, 2013. (978-90-444-1213-0)

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

ISBN: 978-90-444-1213-0

(Eindverslagen Stan Ackermans Instituut ; 2013/031)

Partnership

This project was supported by Eindhoven University of Technology and ASML.

Foreword

The existing framework supporting exposure calibrations is a mature engine. It supports more than a hundred software clients which are used to calibrate the machine and qualify its performance. Being able to support a variety of calibrations and machine configurations makes this engine a powerful tool, but its design imposes a problem: adding new features can break its clients.

Frequently, changes are made to the existing exposure framework to support a new configuration or to support a new feature specific to a calibration step. Current design is not able to isolate functionality that is only for a small subset: this makes the current framework not generic but rather all-supporting.

Despite its age and its value to ASML we have decided to start an investigation to create a new design that is truly generic and scalable but, most important, significantly reduces development effort (progression and regression) on its clients and maintenance effort on the framework itself.

Freedom of thinking to look beyond the everyday solutions is needed, so we decided to ask for the expertise of the TU/e. In advance we knew this could be difficult since usually new Metrologists need over a year to get acquainted with the domain and way of working, yet this project needed to be done in nine months. The combination between the knowledge experts within ASML, and the design and analysis capacity of Spyridon proved itself in a design and prototype of a new generic exposure framework.

The new framework proved itself to be scalable and maintainable. Also, Spyridon decided to follow a direction that is based on an ASML facility, so integrating it in a final product is possible. Some issues were found because the prototype was the first client of several new facilities within ASML. Over the course of the project Spyridon build up a network of people within the projects of these facilities, discussed the issues found and created working alternatives for the prototype, thereby successfully reducing risks for the realization phase.

Thanks to the contribution of Spyridon ASML has a prototype and documentation for a new generic exposure framework that is to be realized upon the first opportunity. Good thing to know is that this framework will act as the reference design for two other to-be-redesigned frameworks.

P.P.A.A. Peeters
J.J.C.P. Schuehmacher

18 September 2013

Preface

This report offers a detailed account of the graduation project for the Software Technology programme on behalf of the Eindhoven University of Technology and the Stan Ackermans Institute. This project was carried out in ASML, a company that designs, develops, and produces photolithography machines, over a period of nine months from January until September 2013.

The project's goal is to evaluate the author as a software designer, while providing ASML with a modern architecture proposal for the exposure framework that is used by the lithoscanner system. This report contains the design solution as well as a description of the process that led there. Therefore, in addition to the new design, the domain, project management conclusions, and retrospective are explained in corresponding chapters.

Spyridon Strouzas
18 September 2013

Acknowledgements

This project could not have been completed if it were not for the help of a large group of people. Therefore, I would like to thank all people that have contributed to my project and I would like to take the opportunity to thank the following people explicitly.

Within ASML I was in very close contact with my supervisors. I want to thank Patrick Peeters, Jelle Schuehmacher and Mamoun El Ouasdad for providing continuous support, feedback, and guidance throughout the project. They assisted me in getting familiar with the ASML domain and also in developing personally as a software engineer. I would also like to thank Ed de Gast, my group leader, for his contribution in the project and the feedback, especially near the end.

More on the technical side of the project I would like to thank Tycho Hilhorst and Peter Barna, the people directly related to the Generic Exposure Framework, for their assistance. Also my gratitude goes to Danny Handoko, Tanja Gurzhiy and Oleksii Bidiuk who provided valuable help and feedback regarding the use of the Viper framework.

From the university side I am grateful to my supervisor, Harold Weffers, for the support and steering he provided in the duration of this project. I also would like to thank the director of the PDEng Software Technology program, Ad Aerts and the scientific director Johan Lukkien for giving me the opportunity to be part of the program, and the program's secretary, Maggy de Wert, for assisting with all the issues regarding me and my colleagues for the past two years.

I would also like to thank all my Software Technology colleagues for all the feedback, support and nice memories during our time together.

Additionally, I would like the express my gratitude to my parents, my sister and my friends in Greece for their support regardless of the distance.

Finally, I want to thank all others that I failed to mention for any kind of contribution throughout the nine months of the project.

Spyridon Strouzas
18 September 2013

Executive Summary

The Generic Exposure Framework provides all the necessary functionality to Calibration, Performance and Diagnostic (CPD) tests when exposing a test wafer is required as part of the system calibration sequence. The Generic Exposure Framework exposes to CPD tests a set of options that can be set depending on different needs. Then, transparently to the CPD test, it invokes all the necessary subsystems for the exposure and creates the expected output in the form of a Diagnostic Report file (Test Log).

This framework is very crucial for the optimal function of the machine in the customer's factory and has been part of the software for more than 13 years. Because of its current design, changes that are frequently required, introduce a significant effort. Thus the decision to start an investigation for its improvement has been taken.

The main issues of the current Generic Exposure Framework are the following:

- The interface towards its clients is large and is changing often. This has a very large impact because it affects all 200 clients using it.
- The UI provided by the framework allows very few customization features and is based on a deprecated template.
- The level of complexity of the implementation is very high. In addition, the lack of people with knowledge of the framework makes it unmaintainable and hard to work with.

To tackle these issues a new software design for the Generic Exposure Framework was created. This new design is based on Viper, a python-based ASML-specific framework that is the current accepted way of working for CPD tests. The design's feasibility was one of the main questions for this project and it was answered with the creation of a prototype. The main functionality of the framework was tried out and verified through this prototype.

The new design provides the same functionality as the existing framework and puts more emphasis on the qualities that make it easier to work with, to understand and to maintain. It offers:

- A flexible interface that reduces the impact in the case of change. This provides the degree of elasticity in order to reduce effort in the case of interface modification.
- A customizable UI per client through a very intuitive and easy to use tool. This offers the amount of freedom that the clients require.
- A far smaller and modular resulting component. There is separation of concerns due to the object oriented design and the layered architecture followed.
- The use of the new Viper framework. This means that the Generic Exposure Framework follows the up-to-date ASML way of working.

This project was the first feasibility attempt that opens the road towards the introduction of a new design for the Generic Exposure Framework. It represents the initial approach to create a high level solution that addresses the most critical issues of the current framework. Since this project proved that the proposed solution is feasible, I recommend a follow-up to establish the implementation details and create the component for integration with the rest of the ASML software.

Table of Contents

List of Figures

List of Tables

1.Introduction

Abstract – In this chapter lies the introduction of the context in which the Generic Exposure Framework is used. The role and purpose of ASML is briefly explained alongside with the part that photolithography plays in the chip manufacturing process. Also the outline of the report is presented.

1.1 Context

ASML is the world's leading provider of lithography systems for the semiconductor industry, manufacturing complex machines that are critical to the production of integrated circuits or chips. Investing heavily in research and development, ASML is now one of very few companies in the world capable of developing the next generation of chip-making machines that will allow Moore's Law to continue. This leads to even smaller, cheaper, more powerful and energy-efficient semiconductors.

Lithography is the most important and critical part of the semiconductor production process:

- It determines how much circuitry can be packed onto a chip — controlling the size and shape of all chip components, connections and contacts.
- It is used in making each layer of the chip — typically, that's about 30 steps to selectively grow, modify and etch out the features in every chip.

In electronics, 'photolithography' or 'nanolithography' is imaging a picture of electronic circuits onto a light sensitive layer of a silicon wafer. This wafer will eventually contain a number of chips. Today's chip lithography uses ultraviolet light to shrink complex patterns and print microchips with features down to 40 nanometers at a very high speed. That makes lithography the most advanced, costly and demanding technology in chip making.

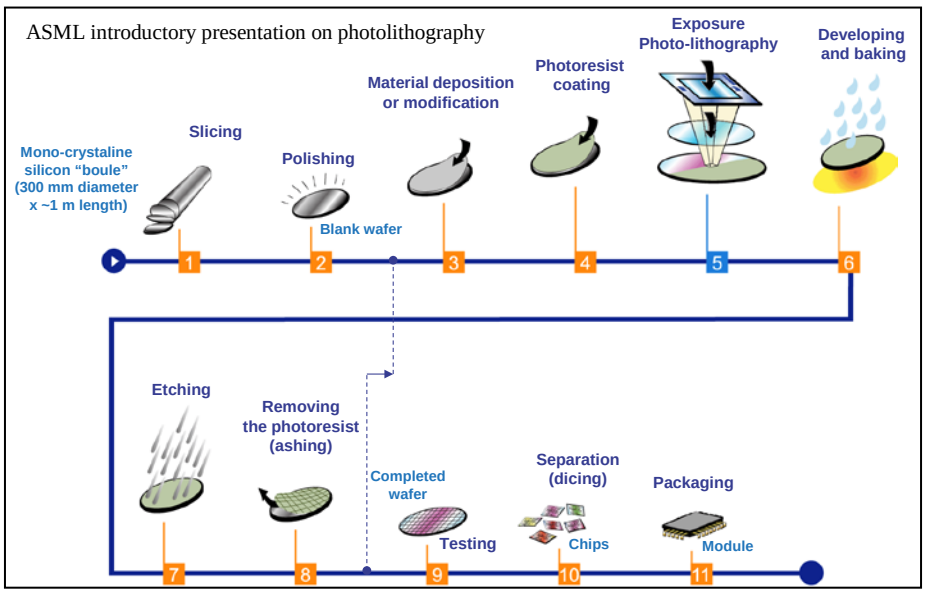

Figure 1 - The manufacturing sequence

The semiconductor manufacturing process (Figure 1 - The manufacturing sequence) contains a number of steps. Everything starts with a raw block of silicon and ends with ready to deploy chips. The steps between 3 and 8 are repeated multiple times (20-30) before the wafer is completed and the chips can be sliced. The details of each step are:

- Material deposition or modification – The wafer is cleaned from undesired contaminators. Then it is heated to remove any moisture. A liquid is applied to make the surface water repellent.
- Photoresist coating – The wafer is covered with unexposed photoresist.

- Exposure photolithography – Using light to make a chip pattern on the photoresist.
- Developing and baking – Leaving the chip pattern as a photoresist mask on the surface of the wafer.
- Etching – Removing silicon and other unprotected by the resist material mask – defined by the chip pattern – using chemicals or plasma.
- Removing the photoresist – Removing the resist mask using plasma.

ASML is creating systems that are responsible for the most demanding step of the sequence, the Exposure photolithography. It is based on the principle of optical lithography (Figure 2).

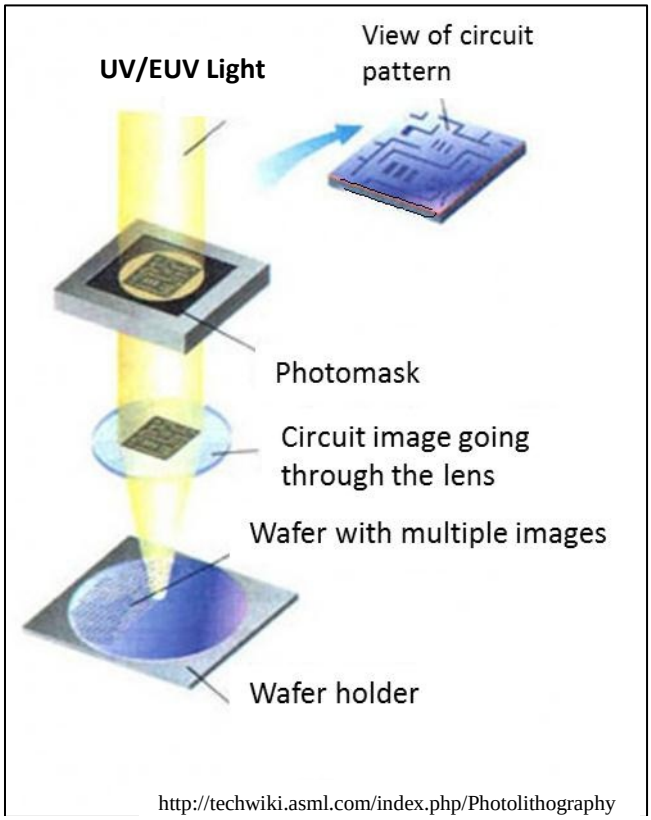

Figure 2 - Optical lithography

In this step the system automatically exposes a predefined chip pattern upon a wafer at certain positions. The chip pattern to be projected on the wafer is drawn on a transparent photomask. The photolithography system shines UV or EUV light through the photomask, projecting a shadow of the chip pattern on the wafer. The photoresist reacts to the light. The parts of the photoresist that react harden and protect the areas directly beneath, allowing everything else to be etched away.

The main goal of the ASML system is to have as high as possible wafer production volume while maintaining the quality and accuracy of the exposures. The quality and accuracy of the exposures is achieved by calibrations that adjust the right parameters in the system. The *Generic Exposure Framework* is part of this important calibration process by providing the exposure capability to calibration software. Currently it suffers from a number of issues. Its improvement is this project's goal.

1.2 Outline

This report is organized in the following chapters

- Chapter 2 presents the identified stakeholders. The main stakeholders for this project were identified in the early phase of this project based on different points of interest for the Generic Exposure Framework.

- Chapter 3 describes the context around the Generic Expose Framework and important information about the Framework itself. This information aims to provide the terms that will give the reader a better understanding of the rest of the report.

- Chapter 4 gives the overview of the problem at hand, a more detailed analysis of the situation and the expected outcome of this project.

- Chapter 5 presents the feasibility analysis of the problem at hand. It shows the relation to the challenges and risks that were identified in the early stages of the project.

- Chapter 6 shows the process used to gather the project's requirements. Following this, the important scenarios that concern the Generic Exposure Framework and the functional and non-functional requirements are described. Finally, the identified design competencies that are important for the success of this project conclude this chapter.

- Chapter 7 discusses some important design decisions. These decisions are the foundations of the new GEF architecture. The usage of other existing frameworks as well as the implementation language decision can be found in this chapter.

- Chapter 8 describes the overall architecture of the new GEF. The two main parts are presented, the internal structure and the interface. The important characteristics of the internal structure of the new GEF design can be found in Section 8.2. The characteristics of the interface are described in Section 8.3.

- Chapter 9 provides a more detailed look at the GEF internal design. In Section 9.1 the high level view of the design is given. In addition the layers of the new framework and their responsibility are explained. The client part is explained in Section 9.2. In Section 9.3 the sequence of the execution can be better understood. Furthermore, the sequence diagram there makes the interaction of GEF-related components more clear.

- Chapter 10 describes the implementation approaches of the design during this project. The role of the implementations was mainly to prove the feasibility of the design and to discover pitfalls that would require additional effort. Feasibility implementations were used for decisions in the early stages and prototypes for the later stages of the project.

- Chapter 11 shows the capability of the new design to provide the expected functionality. The two most important outputs of GEF are compared between the original version and the prototype. The comparison shows that the prototype can successfully create the expected output and fulfill its role in the calibration sequence. Also the realization of the requirements of Chapter 6 by the prototype is shown.

- Chapter 12 shows the results of this project. The two main achievements are the new design proposed for GEF and the prototype that proved its feasibility. Furthermore, the future steps that ASML can follow as a consequence of this project are mentioned.

- Chapter 13 presents an overview of the project management techniques used in this project. The initial planning and how it evolved is also explained. Finally, a list of potential risks for the project is shown along with the mitigation strategies that could be applied.

- Chapter 14 offers a reflection on the project, looking back into what proved to be good practices and what could have been improved. Furthermore, the design competencies are revisited and their role on the outcome of the project is reexamined. ■

2.Stakeholders

Abstract – In this chapter the identified stakeholders are mentioned. The main stakeholders for this project come from ASML and in coordination with them the project's requirements were defined.

Table 1- ASML stakeholders

Name	Role	Responsibility
Patrick Peeters	ASML Project Supervisor - Functional Cluster Architect	• Providing architecture vision and high level information. • Overlooking the progress of the project and offering guidance. • Having responsibility for the part of the software that the Generic Exposure Framework belongs.
Jelle Schuehmacher	ASML Project Supervisor - Metrology department S/W engineer	• Providing crucial domain specific information. • Overlooking the progress of the project and offering guidance.
Jan Portegijs	Project Creator – Previous owner of the Generic Exposure component	• Providing information about the Generic Expose Framework and rationale behind the project.
Tycho Hilhorst	Current owner of the Generic Exposure component	• Ensuring that the project meets the vision and roadmap of ASML for the Generic Expose Framework.
Peter Barna	Current owner of the Generic Exposure component	• Ensuring that the project meets the vision and roadmap of ASML for the Generic Expose Framework.
Danny Handoko	Calibration and Performance Framework (Viper) lead designer	• Providing information on the use of the Calibration and Performance Framework in the project.
Ed de Gast	ASML group leader from the Management Department	• Ensuring that the project results meet the ASML standards.
Edwin Linschoten	Project Lead in Metrology Image Alignment group	• Being a possible user of the project's result.
Pieter Goudzwaard	Functional Cluster Architect	• Being a possible user of the project's result.
Stanislau Shumiacher	Metrology Image Alignment group	• Providing technical information because of experience and future use of the Generic Exposure Framework.

3.Domain Analysis

Abstract – In this chapter, the context around the Generic Expose Framework and important information about the Framework itself are described. This information aim to provide the terms that will give the reader a better understanding of the rest of the report.

3.1 Metrology

The Generic Exposure Framework is part of the calibration process of the ASML system. This calibration process ensures that the system is in optimal condition when used for production. This is the responsibility of the Metrology domain. The main goal of Metrology is to ensure that the exposure of a chip pattern onto the wafer has minimum focus and overlay errors.

The image focus errors are related to the lens effect that is used to scale down the size of the chip pattern. In ASML, the light after crossing the pattern goes through multiple lenses before ending up on the wafer. These lenses scale down the size of the pattern in order to bring it to the desired scale. The correct alignment of these lenses and their distance from the wafer is crucial for the image to be projected accurately. Keeping the focal point of the lens as close as possible to the wafer surface ensures that the sharpest circuit image is exposed (Figure 3).

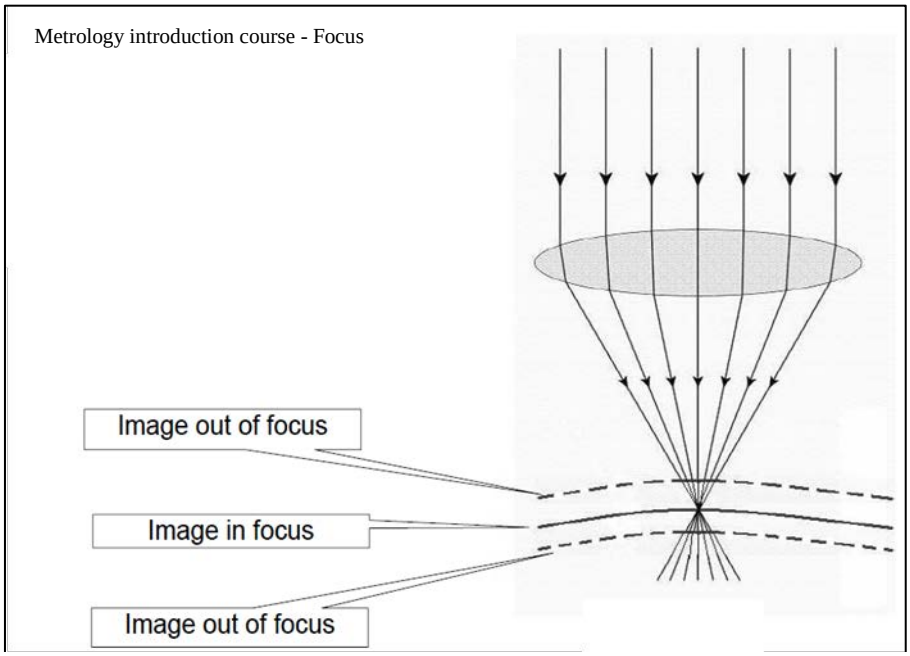

Figure 3 - Focus

The overlay errors are related to the accuracy that two layers are exposed on top of each other. Each wafer is exposed multiple times before it is ready for the next phase of the manufacturing sequence. The final product has several layers of features that contribute to the intended functionality (Figure 4).

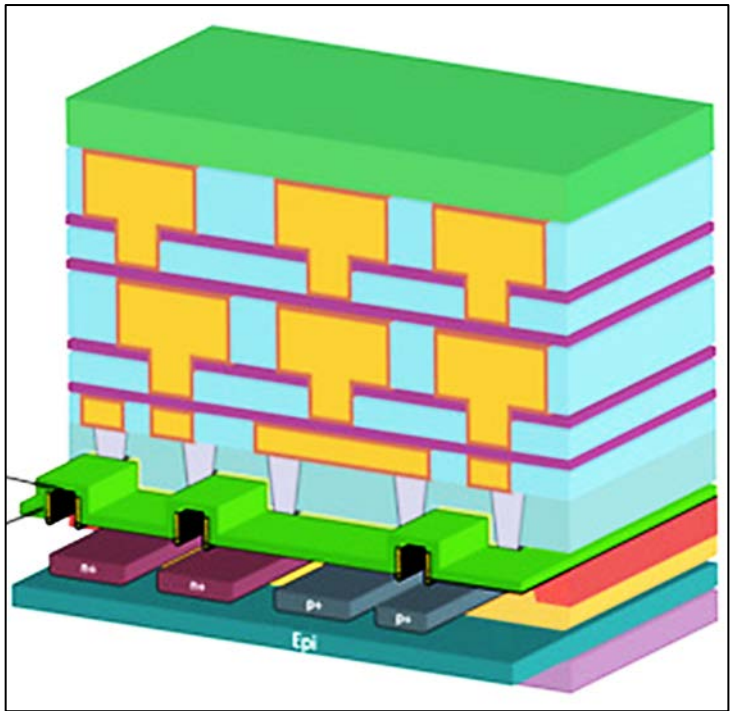

Figure 4 - Ideal Overlay of Layers

Better focus and overlay allows smaller and faster transistors, higher memory density and cheaper chips. Metrology achieves this by performing regular calibrations on the system during its lifetime on the customer side. The result is better focus and overlay by adjusting system parameters. These calibrations are realized by software called Calibration, Performance and Diagnostic tests.

3.2 Calibration, Performance and Diagnostic Tests

Calibration, Performance and Diagnostic (CPD) tests are performed in multiple phases during the system's lifecycle:

- Build-up (system development)
- Factory acceptance tests (before shipment)
- Site acceptance tests (at the customer site)
- Production (during the wafer production process)

CPD tests automate multiple calibration procedures and lead to the fulfillment of

- Performance specifications
  - Compensate for mechanical/electrical imprecision.
  - Compensate for the impact of wear and environmental influences on the system.
- Targeted production time
  - Reduce down time (scheduled and unscheduled)
  - Reduce Mean Time To Repair (MTTR)
- Volume production
  - Reduce lead time in factory
  - Improve man-machine ratio
- Improved development
  - Gather machine knowledge for diagnostics

The CPD tests can be categorized in three calibration groups, depending on the frequency and duration of their execution.

3.3 Calibration Categories

There are three types of calibrations:

- Inline calibrations
- On-line calibrations
- Off-line calibrations

Inline calibrations are automatic calibrations that have a small duration and are performed at a high frequency. They occur without the operator's interaction, before every wafer exposure or even more often.

On-line calibrations are also automatic, require more time than the inline calibrations and occur 3-4 times a day. Both inline and on-line calibrations are performed during the normal operation of the machine without stopping the wafer production process.

Finally, off-line calibrations are manual calibrations that require the largest amount of time. They are performed during the assembly of the machine in the ASML factory and at the customer side. They are necessary for the adjustment of the machine before the start of the production process. At this point, important parameters of the system are adjusted based on testing sequences.

An important off-line calibration sequence (Figure 5) involves the following steps:

1. Expose a test pattern on a wafer
2. Read the wafer after the exposure
3. Adjust system parameters based on the expected and actual results on the wafer

Each step is performed by different CPD tests. There are multiple CPD tests for the exposure, read and adjustment step. The *Generic Expose Framework* is responsible for assisting CPD tests to perform the first step.

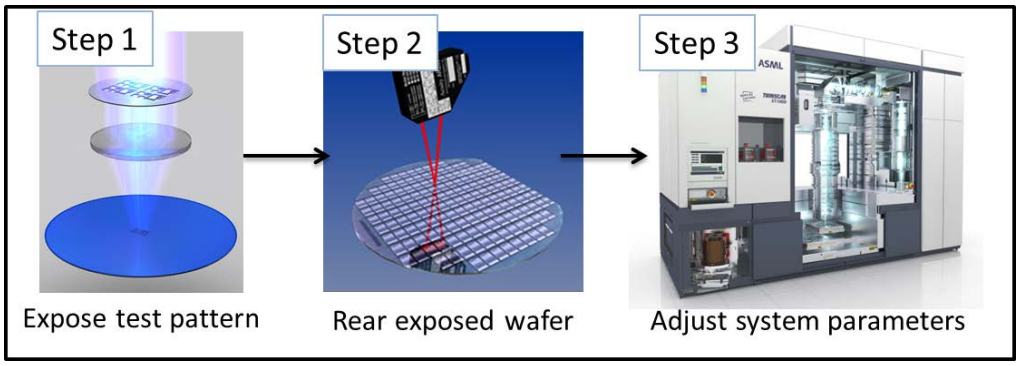

Figure 5 - Off-line calibration sequence

3.4 Generic Exposure Framework

The exposure of a chip pattern onto a wafer is the main goal of the ASML system. To achieve that, the cooperation of all parts of the system is required, making the exposure the most complicated action. In order to perform an exposure, multiple components of the machine need to cooperate, which means most of the software and hardware has to perform the appropriate actions. The normal production sequence has a certain way of setting up the machine for high volume wafer exposure.

On the other hand, CPD tests that follow the sequence of Figure 5 require exposure of smaller number of wafers for diagnostic purposes. Thus, they need to be more flexible when it comes to what they expose to the wafer with options that are specific for calibration. CPD tests that belong to Step 1 have the same main goal which is to expose a test pattern on the wafer. On the other hand, they differ from each other when it comes to settings and parameters for the exposure. The normal production sequence cannot be used for this scenario.

This is why the Generic Exposure Framework was developed. Its main responsibility is to offer the exposure functionality to the multiple CPD tests that need to expose a test pattern. The complicated and common task of preparing the system for exposure and coordinating all the necessary components is provided by the Generic Exposure Framework, while CPD tests only need to set the test-specific parameters.

The important functionalities that the Generic Exposure Framework offers to its clients (CPD tests) are:

- **The exposure production engine**
  By offering the functionality of the exposure engine that is used on the high volume production of wafers, the exposed test patterns from the CPD tests represent the real exposure behavior and performance of the system. In this way the calibration sequence works with data similar to the production sequence.
  CPD tests can adjust settings and parameters of the exposure engine in order to test the behavior of the system under specific conditions.
- **Extension points for test-specific behavior**
  While the exposure sequence is common between all of the framework's clients, there are some points that allow each test to inject extra functionality. This allows the Generic Exposure Framework to be more flexible and fit to its client's needs.
- **Test Log creation after exposure**
  Since the exposure of a test pattern is the first step of the calibration sequence, there must be some connection with the next step. This connection is the exposed wafer and a Test Log that contains the information about the settings and parameters of the exposure. This Test Log is used by the third step that does the adjustment of the system.
- **UI for settings and progress updates**
  The clients of the Generic Exposure Framework can use the user interface that is provided by the framework and adjust it to their own needs. The layout is very similar to the real production UI. ■

4.Problem Analysis

Abstract – After presenting the project's domain and context, the problem that needs solving can be better understood. This chapter provides the overview of the problem at hand, a more detailed analysis of the situation and the expected outcome of this project.

4.1 Problem Statement

The Generic Exposure Framework is part of the ASML system software for a large period of time. This framework was used when the first Twinscan system was introduced in 2000. Even then, it was copied and re-used from the older PAS system. This makes the Generic Exposure Framework more than 13 years old.

Evolution in ASML system software is inevitable. New features and requirements of the system translate into changes in the software. This nonstop change towards improvement has an even greater impact on the Generic Exposure Framework. The reason for that is the large and continuously growing number of CPD test clients depending on it. New clients are a result of new requirements for the system that call for changes in the Generic Exposure Framework.

The constantly increasing number of clients is the reason that the issues of GEF are becoming more and more significant and obvious. GEF suffers from maintainability and extensibility issues partially because of its design and partially because of the use of out dated techniques and technologies. Since its creation, many new methods and technologies in software in general as well as in ASML have been introduced. Object oriented languages have been adopted more and more as they offer increased testability, code re-use, extensibility, maintainability and modularization. This makes the procedural implementation of the Generic Exposure Framework appear rigid and harder to maintain.

It has become ASML's intention to replace the current design with a new one that will be taking into consideration the modern methods, techniques and best-practices while tackling the issues that it suffers from. It will also consider the large number of clients and the framework's role within the system's software environment.

4.2 Analysis

In order to come up with a new design for the Generic Exposure Framework, the exact reasons behind its replacement needed to be investigated and defined. This was achieved by analyzing the current design and having frequent discussions with related stakeholders and domain experts. Several requirement meetings were held in order to define what the issues of the current component are. The aim was to answer the following questions:

- Why is the current design problematic?
- Which are the most important issues?
- Which scenarios depict best the magnitude of the problem?

The answers to these questions led to the creation of the problem baseline. The issues that need to be dealt with the new Generic Exposure Framework design are the

- Rigid interface towards the clients
- Limited UI options for the clients
- Out-dated, undocumented and complex procedural implementation

4.2.1. Rigid Component Interface to the Clients

As already mentioned, the Generic Exposure Framework has a large number of clients. These clients are CPD tests that need to use the provided functionality in order to expose a test wafer with some test-specific options.

The interface of the Generic Expose Framework provides to the clients a large data set with options that must be filled according to the needs of each test. Each test is required to fill this data set at the start of the sequence.

After analysis of the interface's exposed symbols (variables), two observations can be made:

1. There are symbols used by a large number of clients. These symbols represent the **common functionality** of the Generic Exposure Framework used by almost all of the 200 clients.
2. There are symbols used only by a handful of clients (1-5). These symbols represent **client-specific functionality** that was added to the interface in order to satisfy particular clients.

The interface has grown over the years with new symbols to support new client requirements. This has an effect on all the existing clients. Due to the rigid and inflexible interface, every symbol addition and modification requires that all the existing clients are tested and possibly adjusted to continue performing as expected. As a result, the required effort for changes in the Generic Exposure Framework's offered functionality is becoming overwhelming.

4.2.2. Limited UI Options to the Clients

The Generic Exposure Framework implements the user interface that the clients are using during the sequence. Clients set the specific settings that are then used to compose the layout of the test's user interface.

The UI implementation is based on an outdated ASML template that is now deprecated. It requires a lot of handwritten code and clients need to define from a fixed list of widgets which should be visible and which not. The layout of the UI and the widgets within are predefined. The client cannot customize the position of the widgets; nor add and remove additional client-specific ones.

The current UI implementation poses difficulties for future requirements of the Generic Exposure Framework's clients. The fixed list of widgets limits the options that new clients can have on their UI and creates complications to future needs.

4.2.3. Out-dated, Undocumented and Complex Procedural Implementation

The implementation of the Generic Exposure Framework dates back to the old PAS system software. This was copied to the later Twinscan system in 2000. Since then, there have been vast changes in techniques and technologies that are supported and promoted by ASML's software guidelines. The Generic Exposure Framework does not fully comply anymore with those guidelines.

Another issue that makes the current form of the Generic Exposure Framework problematic is the poor documentation of its contents and functionality. In addition, the unavailability of the original creators, the complex implementation and the many changes over the years have led to a component that is hard to understand and maintain.

4.3 Project Goal

The problems mentioned previously reveal the reasons behind the need for a new version of the Generic Exposure Framework. The main goal of this project is to provide the software design of this new version. The design must be based on a more modern approach to ensure increased lifespan, in accordance with the ASML software guidelines and with consideration of future changes and needs. Furthermore, it must make sure that the problems of the existing design are dealt with and that the effort to create the new Generic Exposure Framework is outweighed by its benefits. Finally the new design must be implemented in a prototype to showcase in practice the improvements to the existing one as well as the applicability and feasibility of the new techniques used. ■

5.Feasibility Analysis

Abstract – In this chapter, the feasibility analysis of the problem at hand is presented. It is highly dependent on the challenges and risks that were identified in the early stages of the project.

5.1 Issues and Challenges

Even before the beginning of the project, particular issues and challenges were already expected. The creators of this project in ASML had foreseen them and were mentioned in the description and as well as discussed during the interview. Alongside those, a few more would become obvious from the early phases of the project.

5.1.1. Domain and Process Familiarization

The ASML domain is immense. Normally, employees require a few months to become familiar with the context, the tools, the processes and the way of working. This was even more imperative for this project since it involved software that is part of the actual system software used on the lithoscanner. This means that the result must conform to all the ASML standards and follow the predefined procedures.

The author was well aware of this from the beginning. This is why a strong effort was made from early phases of the project to reduce the time required to reach a certain level of understanding and familiarization with the domain and the required processes. The communication part played a crucial role in the effort to extract the much needed knowledge from the stakeholders and domain experts.

5.1.2. Communication with Multiple Stakeholders

The component that is redesigned in this project is a framework. As a result, there are numerous clients using it. Additionally, there are multiple components that provide particular functionality to the framework. This creates a large network of dependencies and requires constant communication with multiple stakeholder types. Decisions concerning the new design of the framework have an impact on this network and consequences for the other components in it. Thus, close contact with all those stakeholders is essential.

5.1.3. Unavailability of Component Expert

The Generic Exposure Framework has been in the ASML system software since early 2000. This means that, in the time span of 13 years, people who have worked on it have moved to other positions within the company or outside. Ownership of this component has changed multiple times and the current implementation is a contribution of numerous developers that have been adding and updating functionality throughout the years. In essence all these issues mean that there is no local expert of this component. Different people know different parts and no one has a complete picture of the Generic Exposure Framework. It is, in other words, legacy code. This was handled by contacting people that had worked on the component the past years and creating a name list to refer to in case of need.

5.1.4. Poor and Out-dated Component Documentation

The very first day, the documentation available on the Generic Exposure Framework was obtained. This was the starting point towards understanding the component and ultimately the source of the problems. Soon it became obvious that the existing documentation was not as detailed as expected and was also not in sync with the actual implementation. Changes were not always reflected on the corresponding document and additional effort was required to obtain a deeper knowledge of the component.

5.2 Risks

Certain risks that could jeopardize the outcome of the project were identified early on. The most important ones are mentioned along with the mitigation strategy applied.

5.2.1. Applicability of Design

The main goal of the project is to provide a new design for the Generic Exposure Framework that tackles the issues of the current design and preferably uses the latest available technologies and techniques. This means that the author would be the first adopter of more than a few technologies that were recently introduced within ASML. The combination of new and old technologies is likely to cause incompatibilities and issues. Such issues were to be mentioned in the design document for ASML. These would be used for consideration and future improvement. For the prototype, in case of these issues, stable and already used versions would provide a functional alternative for these parts.

5.2.2. Carrier Project to Integrate the Design

This project is considered a feasibility study. The applicability of a new and modern Generic Exposure Framework has to be evaluated. The eventual implementation of the new design in the lithoscanner software is only feasible if the design is accepted and incorporated in a carrier project. This is why the project needs to be made known to project leaders who can incorporate it and launch it. A significant effort was made to identify and keep this group of people involved.

One important condition for adaptation of the new design by a carrier project is that the new design's benefits outweigh the implementation effort. This applies for the implementation phase. In the long term the maintenance effort of the new design must also be lower than the effort that is put on the current design.

5.2.3. Evolutionary Development

The current component has gained more and more clients over the years. New clients get introduced almost every year and modifications are then required. This characteristic makes the component relatively unstable and frequently altered. The new design must have a way of handling this change rate or else it will not be able to endure.

5.2.4. Implementation Detail Level of the Prototype

The implementation of the existing component is considered complicated and relatively large. The estimations that ASML has for a complete reimplementation of the component by ASML experts exceeded the duration of this project. Thus, for the prototype that was expected as one of the deliverables, the level of detail and the functionality provided had to be agreed upon. Frequent meetings were organized to discuss the prototype's feature list and final expectations. ■

6.System Requirements

Abstract – In this chapter, the process used to gather the project's requirements is presented. Following this, the important scenarios that concern the Generic Exposure Framework and the functional and non-functional requirements are described. Finally, the identified design competencies that are important for the success of this project conclude this chapter.

6.1 Requirements Gathering Process

In the beginning of the project, a list of people that would be involved had to be established. In order to create this list, some white board meetings to be used as brainstorming sessions were organized with the help of the ASML supervisors. People who had any kind of interest or relation with the Generic Expose Framework were invited to discuss their ideas about it. The goal was to have as many people as possible and eventually narrow the list down to the most important stakeholders who could provide the best contribution. After these white board meetings, a smaller list was composed with stakeholders who agreed to participate and provide feedback during this project. The stakeholders on the list include

- Former and current component owners of the framework.
- Owners of components who are users of the framework.
- Software architects responsible for the particular software part.
- Developers who have worked with the component in the past.

During the initial investigation on possible solutions, and after some important decisions were made, a few more people were added to the list:

- Designers and developers of components used for the solution.

The listing of people who was composed after this point can be found in Chapter 2.

For the duration of the requirement extraction phase, multiple meetings with the stakeholders were organized. Preferably individual meetings took place in order to gather as much information as possible in the area of expertise of each person. The goals of these meetings were to

- Create a first version of a requirement list that could be reviewed by all stakeholders and lead to further refinement and discussion.
- Increase the familiarity between the author and the people. This would lead to more information sources and assistance during the project and also higher chances of adoption of the final solution.
- Establish a better understanding of the context in which the Generic Exposure Framework is used from different points of view.

After this first round of meetings, a requirement list was created. This list included most of the requirements that were mentioned during the meetings. The idea was to elicit a fruitful review session of this list and end up with those items that were most important. The list was distributed to all stakeholders and a meeting was planned to discuss the outcome of the review. The review meeting provided some interesting discussions and eventually led to a globally accepted and stable requirement list.

The main aim of this phase was to create a requirement list that would satisfy the most important stakeholders and allow the author to move forward to the design phase of the project. The list was not meant to be final but relatively stable, since the research nature and duration of this project were expected to cause changes. The non-functional requirements were given more emphasis during this phase. The important problems of the current design are caused by the lack of particular non-functional requirements, so these should definitely be covered by the new design. Additionally, the new design is based on satisfying these non-functional requirements, while most of the functional requirements can be refined later during the iterative prototype phase.

6.2 *Important Scenarios*

For the complete understanding of the component, the scenarios under which is being used had to be identified. The scenarios provide a different perspective and reveal what functionalities are considered more important. The Generic Exposure Framework can be seen from the point of view of two major actors:

- ASML S/W developer
- Machine Operator

These two actors perform the major scenarios of the Generic Expose Framework that portray the usage context of the component.

6.2.1. ASML S/W Developer

An ASML developer has two main scenarios in which he interacts with the Generic Exposure Framework:

- Creation of a CPD test with wafer exposure requirements that belongs to the calibration sequence shown in Chapter 3.
- Implementation of an update or change to the Generic Exposure Framework.

These two scenarios are shown and described in Figure 6 and Table 2.

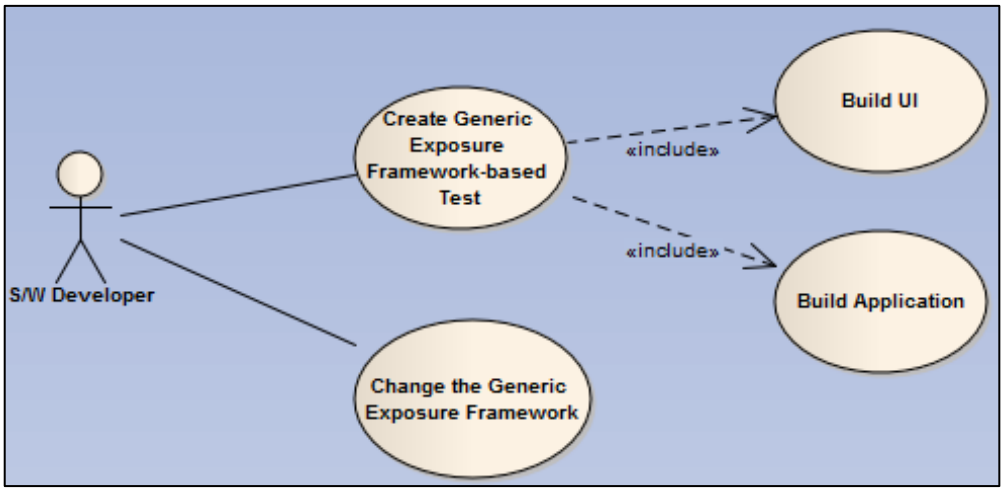

Figure 6 - S/W Developer scenarios

Table 2 - S/W Developer scenarios description

Actor	Scenario	Description
S/W Developer	Create Generic Exposure Framework -based test	A software developer creates a new exposure test based on the Generic Exposure Framework. He specifies the test-specific functionality that is going to be used along with the common framework functionality. It is decomposed to building the UI of the test and building the application with the test data and logic.
S/W Developer	Change the Generic Exposure Framework	An update is needed in the Generic Exposure Framework and the software developer needs to carry out the changes in the component. This occurs either when new functionality is needed by a test, when a change occurs in one of the components used by Generic Exposure Framework or a bug fix is required.

6.2.2. Machine Operator

A machine operator interacts with the UI that the CPD test provides. This UI is composed by the Generic Exposure Framework. As mentioned earlier, the exposure of a test pattern on a wafer, that the Generic Exposure Framework is responsible for, is part of the calibration sequence. Thus, the UI of the first step of the sequence is created by the framework and is important that it follows the standards for operator-machine interaction. The scenario is shown and described in Figure 7 and Table 3Table 6.

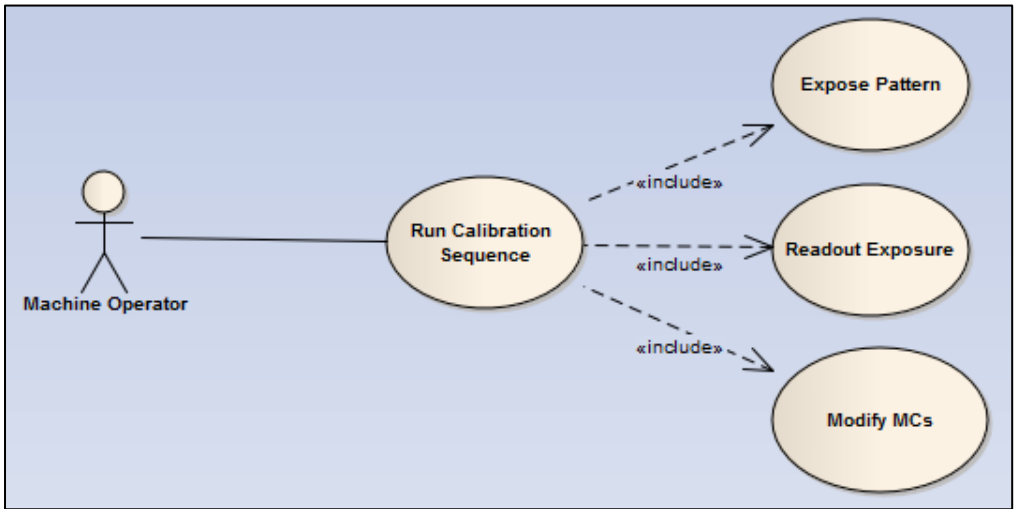

Figure 7 - Machine Operator scenario

Table 3 - Machine Operator scenario description

Actor	Scenario	Description
Machine Operator	Run Calibration Test	The operator of the system runs a calibration sequence. First, an exposure test is executed that is based on the Generic Exposure Framework in order to expose a test pattern. Then, a readout test to read the exposed wafer(s) and finally a test to modify accordingly the machine constants are executed.

6.3 Requirements

At the end of the requirement phase of the project, a stable list of functional and non-functional requirements was composed. The two categories of requirements are shown and explained in the following sections.

For the sake of clarity the GEF abbreviation is used to refer to the Generic Exposure Framework.

6.3.1. Functional Requirements

At first, the functional requirements were collected from existing documentation and the existing implementation of GEF. That list consisted of domain details such as exposure types, exposure modes and settings that GEF had to support and offer. These requirements were considered later on as not beneficial for the new design. They did not have a significant impact in the high level design and could be refined later on. Nevertheless, they were always thought of during the design phase so that they would fit without issues later on.

The functional requirements that were considered very important were those that came from stakeholders during the various meetings. These included additional functionality that was

requested for the new version of GEF and improvements in order to correct certain issues and problems present. The list with the functional requirements is shown in Table 4. The requirements have been grouped into categories. The categories represent parts of GEF that have high importance and need to be improved and that definitely be present in the new version.

The User Interface category contains the functional requirements that are related to the layout of the UI of the CPD tests that use GEF. It defines the creation process and the structure for the visual part of CPD tests.

The Interface for Clients category focuses on the communication between GEF and its clients. It tackles the evolution issues of the interface and the exposed functionality.

The Exposure Test Execution is related to the execution sequence of every CPD test that uses GEF. It defines basic functional features that must be present.

Finally, the Test Log category includes the requirements that are related to the Test Log. The Test Log is the main deliverable of GEF. It is a diagnostic file that contains the input of GEF received from the CPD test and the operator as well as the information and results that were gathered during exposure. This is very crucial for the next steps of the calibration sequence as its contents are then used to modify the machine's constants.

Table 4 - Functional Requirements

ID	Requirement	Priority
	User Interface	
FR1.1	GEF provides a UI mechanism to the client tests.	Must
FR1.2	GEF provides the default/common part of the UI.	Must
FR1.3	GEF has a logical division of UI into tabs with widgets grouped based on functionality.	Should
FR1.4	GEF uses similar parts of the wafer production screen UI.	Should
	Interface for Clients	
FR2.1	GEF provides a common stable interface for test clients.	Must
FR2.2	GEF allows the extension of the interface for test clients with varying requirements.	Must
	Exposure Test Execution	
FR3.1	GEF provides hooks to test-specific functions during execution.	Must
FR3.2	GEF supports input from the UI.	Must
FR3.3	GEF supports input from the test clients.	Must
	Test Log	
FR4.1	GEF is able to create and store Test Logs	Must
FR4.2	GEF is able to add client specific part in the Test Log.	Must

6.4 Non-functional Requirements

As already mentioned the non-functional requirements were given more emphasis during this project. They were deducted by first identifying the problems of GEF and then understanding the underlying cause. The fulfillment of these non-functional requirements by the new design, will address the most important issues that GEF currently suffers from. The non-functional requirements are shown in Table 5.

Table 5 - Non-functional Requirements

ID	Requirement	Priority
	General	
NFR1.1	GEF is a framework for providing wafer exposure functionality to calibration, performance and diagnostic tests.	Must
NFR1.2	GEF uses the wafer production engine, so calibration sequence and production act alike.	Must
	Usability	

NFR2.1	GEF provides a UI with the same look and feel as the rest of the calibration software.	Must
NFR2.2	GEF's new design provides a more convenient and flexible way of creating CPD tests.	Should
NFR2.3	GEF extensions (in functionality and in UI) for new clients have small impact on existing clients.	Should
Documentation		
NFR3.1	Manual for creation of new exposure tests based on the new design.	Should
NFR3.2	GEF's implementation is understandable.	Must
Extensibility		
NFR4.1	GEF allows client CPD tests to extend the common functionality.	Must
NFR4.2	GEF should not have test-specific knowledge but shall be kept generic and unaware of test-specific details.	Should
NFR4.3	GEF must have a decoupled UI from business logic.	Must
NFR4.4	GEF's UI mechanism provides a customizable UI to the client.	Must
Compatibility		
NFR5.1	GEF fits the current software architecture and concepts used in ASML.	Must
NFR5.2	GEF's new design is used for new tests (no backwards compatibility).	Must
NFR5.3	GEF is compatible with the rest of the calibration sequence that sets system parameters based on exposure on a wafer and readout of the result.	Must

NFR1.2: GEF uses the wafer production engine, so calibration sequence and production act alike.
The tests that need to expose a wafer must run in a similar way to the high volume wafer production in order to have realistic exposure results. In order to achieve this, GEF uses a number of other components, including the high-volume wafer production component. This leads to exposures that resemble the wafer exposures that happen during production. As a result, the calibration is more effective and accurate.

NFR2.1: GEF provides a UI with the same look and feel as the rest of the calibration software.
Uniformity is important when it comes to the User Interface. The machine operators must have to deal with a similar user interface for the numerous tests. All these tests must follow the ASML dictated way of UI design. This means that GEF must enforce its clients to have the same layout and only differ in specific details.

NFR2.2: New GEF design provides a more convenient and flexible way of creating CPD tests.
The current version of GEF requires the CPD developers to implement a lot of repetitive code and while at the same time there is not enough freedom and flexibility for the tests. The new design must keep the test creation procedure simple and at the same time powerful so that tests can extend the GEF functionality.

NFR2.3: GEF extensions (in functionality and in UI) for new clients have small impact on existing clients.
An important issue that the current GEF suffers from is that whenever a change in UI structure or in functionality is needed for new clients, then all existing clients are included in the build scope and need to be tested. This creates a large overhead for every change that needs to be done in GEF. The new GEF must be more flexible to changes and not have a big impact on the rest of the dependent components.

NFR3.1: Manual for creation of new exposure tests based on the new design.
One of the issues with the current GEF is the outdated documentation. This must be solved in the new design. Apart from documents for the framework itself, a detailed guide for GEF's clients must be created so that the possibilities and functionalities available for use are shown. Additionally this will serve as a guarantee against misuse of the exposed functionality.

NFR3.2: GEF's implementation is understandable.
Understandability is important at many levels, from individual instructions, functions and up to the overall system design. Through the act of writing intelligible source code, the efficiency of software development can be increased, maintainability can be enhanced and overall productivity can be improved. Writing comprehensible source code gives software engineers an integrated guide to the many details of the system, and best of all, the guide lives in the code itself, through comments, naming conventions and descriptions.

NFR4.1: GEF allows client CPD tests to extend the common functionality.
Software changes and upgrades are occurring regularly. That is why new tests often require extra functionality. GEF must be open for extensions and allow tests to take advantage of new features that are available.

NFR4.2: GEF should not have test-specific knowledge but shall be kept generic and unaware of test-specific details.
The main idea of GEF is to provide the common functionality for an exposure test. The test-specific parts should be provided by the test and be known by the test only. As time went by, more and more test-specific details were implemented in GEF to support some tests. This means that now GEF is contaminated with test-specific parts that are only used by a few tests but known to all clients.
The new GEF should be kept clean of all these test-specific details and provide only common functionality. For this to be achieved, a way of allowing tests to inject custom functionality must be established.

NFR4.3: GEF must have a decoupled UI from business logic.
Currently the UI is part of GEF and the business logic of the framework is closely coupled to it. This makes it hard to change either of them without affecting the other. A more modern approach must be used to separate the concerns of each module. This will make the UI and the business logic easier to change and replace without creating issues for each other. An approach that is based on the MVC (model-view-controller) pattern should be used.

NFR5.1: GEF fits the current software architecture and concepts used in ASML.
GEF is part of the Twinscan software that is executed on the Lithoscanner system. This means that there are strict rules and regulations that must be followed by the component's architecture and design. The ASML software standards must be satisfied and the decisions about the design need to take the ASML context into consideration.
The technologies and mechanisms used in ASML's software are constantly being changed and updated. It is crucial therefore to use the latest possible for the design of the new GEF. This will guarantee the longest possible lifespan for the component.

NFR5.2: GEF's new design is used for new tests (no backwards compatibility).
The change from the current GEF to the new will be quite dramatic and backwards compatibility is sacrificed in the name of a more modern implementation. This means that only new tests that are based on the new design will be compatible. The old tests will still be able to use the existing GEF version.

NFR5.3: GEF is compatible with the rest of the calibration sequence that sets system parameters based on exposure on a wafer and readout of the result.
The Test Log is the main output of GEF and it is passed to the next steps of the calibration sequence. This means that the same Test Log format and structure needs to be used by the new GEF in order to be compatible with the rest of the sequence. So while GEF is not backwards compatible to clients, it is partly backwards compatible because of the use of the same Test Log.

6.5 Design Competencies

As soon as the context and the requirements of the project were defined, several design characteristics that would be important for the new design were identified. Three with high and two with low priority are described in this section. At Chapter 14 the same characteristics are revisited and their fulfillment by the final design is analyzed.

Those that are considered as important for this project are

- **Genericity:** The design focuses on the GEF for this assignment, but it was made clear from early stages that the same design would be important to fit the framework of the second step in the calibration sequence. This design characteristic means that the final solution must ensure that the non-functional requirements are still fulfilled even if the functional requirements change.
- **Realization:** The new design is expected to use technologies and mechanisms that are new in the ASML context and have never been combined before. This highlights the research aspect of the project and the need to answer questions concerning the feasibility and level of complexity of the solution. Additionally, the effort required for the implementation of the new design and integration to the code baseline must be determined. This explains the need for a prototype that will try to shed light on to the most uncertain aspects of the design.
- **Complexity:** As revealed during the requirements extraction phase, one of the issues of the current implementation of GEF is the high level of complexity. This creates a large overhead in understanding the component and implementing changes or updates. Furthermore, being a framework and being used by multiple clients introduces the necessity to provide a clean and straightforward way of use. As a result, the new design must keep the internals of GEF understandable and maintainable and the interface to the clients straightforward and usable. A design and an implementation language that lowers complexity and increases readability must be utilized.

The competencies that were deemed as not highly important for this project are

- **Functionality:** The aim of the design is to prove that the combination of different technologies and techniques is feasible and satisfies the non-functional requirements. Also the level of genericity required makes the specific functionalities less important, since these can be implemented later whenever the decision to integrate the design with the rest of the software is made.
- **Impact:** The context of the project is to allow the author to work without the constraints that typically are imposed on such tasks and freely experiment with a solution that introduces quite a large level of change. Moreover, the decision to not provide compatibility with existing clients shows that impact was not one of the important concerns behind the project. On the other hand, migration to the new design is important. A phased roll-out of the new GEF will allow easier introduction of the new design in the system.■

7.Design Alternatives

Abstract – In this chapter some important design decisions are discussed. These decisions are the foundations of the new GEF architecture. The usage of other existing frameworks as well as the implementation language decision can be found in this chapter.

7.1 Introduction

Early in the project, the expectations and context were defined. The main goal was to investigate and redesign the Generic Exposure Framework given the freedom to think outside the box and without being biased by the ASML way of working. This gave the project an exploratory nature and a broad range of choices. The large number of alternatives would have to be investigated and the most suitable kept in the final solution.

The process of evaluating alternatives and deciding in favor of one was done in two phases. The first phase was more theoretical and included documentation reading, meetings with relevant stakeholders and thinking of applicability of similar solutions from other domains. It usually included small feasibility tests and proof of concepts. The second phase of evaluation was the creation of a large scale prototype that would have some earlier decisions as foundations and incorporate more during the process. Some alternatives were dependent on others, so that pointed the order of the investigation.

In this chapter the decisions that acted as the foundations for the final solution are described. This way the reader can have a better impression and understanding about the rationale behind choices in the final design.

7.2 Use of Other Calibration Frameworks

The current implementation of GEF, as explained in Chapter 3, provides certain functionality to its clients. When GEF was initially designed it was decided that the following must be implemented and provided:

1. Creation of the UI of the test.
2. Exposure of a test pattern on a wafer.
3. Creation of a Test Log file.
4. Creation of a report at the end of the test.
5. Control of the test execution logic.

Points 1, 4 and 5 are generic functionality that is currently required by almost all of the CPD tests in the Twinscan software. As a result, generic frameworks have been developed to support different kinds of CPD tests and provide them with such functionality. The important difference of GEF is that it provides additionally points 2 and 3.

For that reason, it was decided early in the project that existing frameworks could be used in order to simplify GEF and have it provide only the functionality that makes it unique. Creating a framework from scratch was one option but it was eventually rejected due to significantly larger effort required. This would be very beneficial since GEF would become much smaller and easier to maintain and understand. Also use of an existing framework would reduce dramatically the effort needed for implementation compared to a completely new and independent solution.

After investigation the most modern framework for CPD tests in ASML was chosen to cover part of the GEF functionality. This is called Viper and it is a calibration and performance framework based on the programming language Python. Viper is the dictated way to work with CPD tests in ASML. It is a generic framework that is created to cover a wide range of common CPD test requirements. The GEF-specific functionality must be developed separately. Incorporating it in the new design ensures that GEF uses an ASML state-of-the-art mechanism. This way, Viper is accountable for certain functionality and GEF has fewer responsibili-

ties. The Viper framework is the common way of CPD test creation and incorporating GEF into this way of working will make it more accessible and understandable to developers.

The decision to use Viper as a supporting framework is the first major decision for this project and has a high influence on most of the design:

- GEF will be using Viper functionality.
- CPD test creation for GEF will be influenced by the way CPD tests are created for Viper.
- Certain functionality is removed from GEF since Viper will be providing it. Such functionality includes the UI implementation, the creation of the test report and the test execution logic.
- The implementation language of Viper is Python. Therefore, a large part of GEF will be using Python to communicate with Viper. The GEF clients are also going to be implemented in Python to be compatible with Viper.

7.3 Framework or Library

After the choice of Viper as a supporting framework, the way of placing GEF in relation to Viper and the CPD tests had to be decided. In a typical Viper use case, Viper is being used by the CPD test by exposing an interface with the available functions. The CPD test uses these functions in order to create the desired test sequence. This sequence is composed of steps. Each step has a particular task. The steps compose the test sequence and based on that, Viper executes the steps in the defined order. The typical Viper use case is shown in Figure 8.

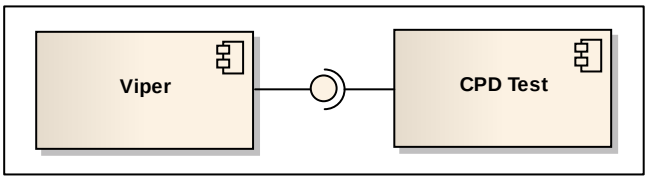

Figure 8 - Typical Viper - CPD Test use case

Now that GEF needs to provide additional functionality, there were two possible alternatives. The first is to implement GEF as a framework that would act as an intermediate between the CPD tests and Viper. The second would be to implement GEF as a library that would expose certain functionality and would be used by the CPD test when needed without GEF having any interaction with Viper.

7.3.1. Framework Alternative

The framework alternative means that GEF is exclusively communicating with the clients. All required functionality from Viper is being called by GEF, and CPD tests have no access to Viper. CPD clients are using the GEF exposed interface to provide the test-specific details for the execution of the test. The high level view of the components is depicted in Figure 9.

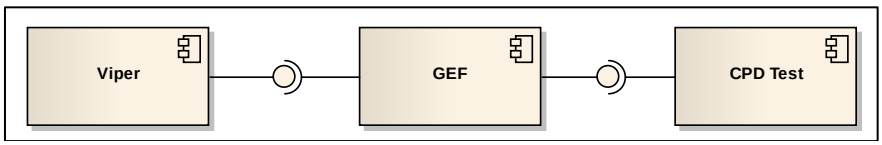

Figure 9 - GEF framework alternative

This design is wrapping the Viper functionality by intercepting the calls from CPD tests. GEF can receive the details from the CPD test, add the common part and functionality that all CPD tests require and then notify Viper to proceed with the test execution.

7.3.2. Library Alternative

The library alternative means that GEF provides functionality that will be used by CPD tests in specific parts of the test execution. The CPD test will be using Viper as in the typical CPD

use case (Figure 8) and in specific points GEF functionality will be called. This alternative is shown in Figure 10.

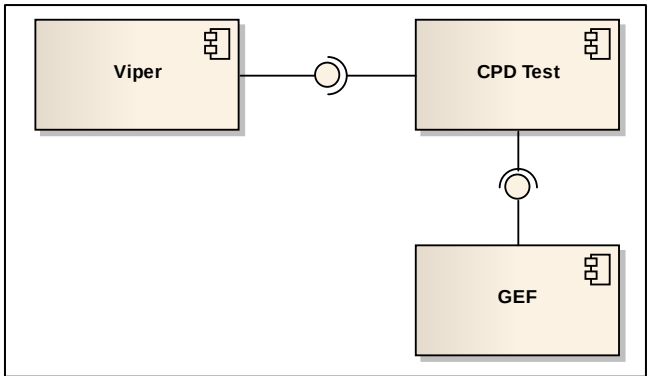

Figure 10 - GEF library alternative

This design makes the CPD in control of the execution and GEF an assisting library by providing the needed functionality.

7.3.3. Conclusion

The main consideration for this decision is the creation of the CPD test. One of the non-functional requirements that must be satisfied by the new design is the level of usability by the CPD tests. GEF must provide its functionality in the most convenient way and facilitate the creation of CPD tests that are simple and effortless. In order to have this quality in the new GEF, the existing GEF tests had to be understood. The important factor to look for is the level of similarity.

After investigating tests of the current GEF, it is relatively clear that the logic behind the tests is the same. The sequence is the same for all of the tests and only specific details change from one to the other. This means that the steps for the test execution are almost identical.

The framework design is more suitable because it offers

- Simple test creation, since the steps that compose the execution sequence are implemented by GEF and the test only provides the parameters.
- Client dependency only on GEF's interface.
- Minimum repetition of identical code between tests since all common part is in GEF.
- Single point of change in the case of update in GEF functionality. Since common parts are in GEF, then a change would only occur there and not in the clients.

If the clients are using different execution logic and steps then a library approach is more convenient and flexible. However, since clients are almost identical, making GEF a framework wrapped around Viper makes simpler the creation of tests than in the library alternative.

To conclude, the framework approach was considered to be more fitting to the requirements of GEF and is the base of the new design.

7.4 GEF Implementation Language

Another effect of the selection of Viper is the introduction of Python in the new design. Viper is based on Python to provide rapid development of CPD tests and to allow the developers to focus more on the functionality than trivialities in the code. Also the resulting implementation is significantly smaller in size and more understandable for newer developers than the C equivalent.

On the other hand, the implementation language of the current GEF is C. The source code of the framework itself, as well as that of the clients is implemented in C. This is expected to

create an extra effort mapping and converting the existing functionality from C to Python. Additionally, complete migration to Python might introduce some unforeseen issues. Particular features that currently are implemented in C might not have direct equivalents in Python.

Translation of the component from C to Python by itself will not be beneficiary if the design itself is not improved. This means that the implementation language is partially the problem. Keeping part of the implementation in C has also benefits, such as reduced migration effort, use of already tried code. With a mix of both languages the design can take advantage of the existing internals while recreating the overall design for better decomposition.

For inter-language communication an ASML-specific language for interfaces is used. This interface language allows the communication of parts implemented in different programming languages. Therefore components are capable of using functionality from other components without being aware of the implementation language. Additionally, this interface language can be used within the same component and allow the use of multiple languages. This makes the design modular and keeps it future proof against introduction of new languages.

In the case of the new GEF design Python is a certainty. As for C, its existence is dependent on two factors:

- Reuse of existing GEF parts.
- Callability of needed components from Python.

In order to have flexibility during the implementation of the new design, both languages are going to be used by taking advantage of the inter-language communication that the interface domain-specific language (DSL) provides. As for the usage of C, both cases were encountered. The use of existing GEF parts contribute to the reduction of the implementation effort and reduced maintenance. Also certain components that are needed by GEF did not support being called from Python. So the use of C was required. Thus, the end result of the implementation is mostly in Python supported by a C part. ■

8.System Architecture

Abstract – In this chapter the overall architecture is described. The two main parts of the project are presented, the internal structure and the interface. The important characteristics of the internal structure of the new GEF design can be found in Section 8.2. The characteristics of the interface are described in Section 8.3.

8.1 Introduction

The architecture of the new GEF was created having in mind mainly the non-functional requirements mentioned in Chapter 6. From those, three proved to have the most significant influence on the final architecture:

- Compatibility with the rest of the calibration sequence.
- Allowing the clients to extend the common GEF functionality.
- Minimizing the impact when clients introduce new functional requirements to GEF.

The first requirement means that the end result of a CPD test must provide the same result as it would with the current design. The last two requirements resulted in an extensive investigation on the interface structure of GEF towards its CPD clients. A more flexible solution was needed.

8.2 Internal Structure

Compatibility to the rest of the calibration sequence essentially stands for the exposure of the test pattern on a wafer and the creation of the Test Log. These two deliverables are used by the rest of the calibration sequence and must not be altered in the new design. The functional requirements of the current design are all applicable and expected in the new one. Consequently, the execution logic of the new design will be dictated by the one in the existing design.

Additionally, in order to fulfill the important non-functional requirements, the new design is based on a layered architecture, combines Python and C and follows the Model-View-Controller pattern.

8.2.1. Sequence Decomposition

The execution sequence of the current GEF is in summary one large sequence that performs the necessary actions for the test. In order to make the new version more modular and understandable, the sequence was broken down into smaller parts called steps. The existing GEF functionality is mapped into specific steps. These are then decomposed into sub-steps and so forth. As a result the different parts of GEF sequence are now decoupled from each other and are more independent. This can be seen in Figure 11.

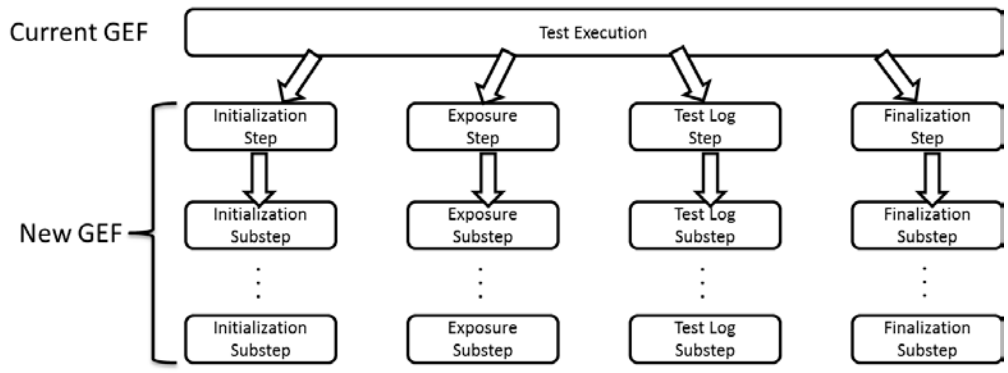

Figure 11 - Decomposition and abstraction levels of execution sequence

This decomposition achieves better modularity that makes the sequence more understandable and the parts changeable and less dependent on each other. Furthermore, it allows what is

described in Section 8.3.2 with injection of test-specific functionality and swapping of step implementations.

8.2.2. Layering

The layered architectural style focuses on the grouping of related functionality within an application into distinct layers. The layers are stacked on top of each other and each layer hosts parts with common roles or responsibilities. Communication between layers is done through defined interfaces. This keeps the layers loosely coupled.

In the new GEF design, the top layer contains the interface of the component. In this layer the wrapping of the Viper interface is done and the combination of test-specific and GEF common parts is performed. The middle layer contains the logic for the creation of the execution sequence. Finally, the bottom layer contains the low level functionality that the sequence calls. The bottom layer is also responsible for communication with other components needed in GEF. This concept is shown in Figure 12.

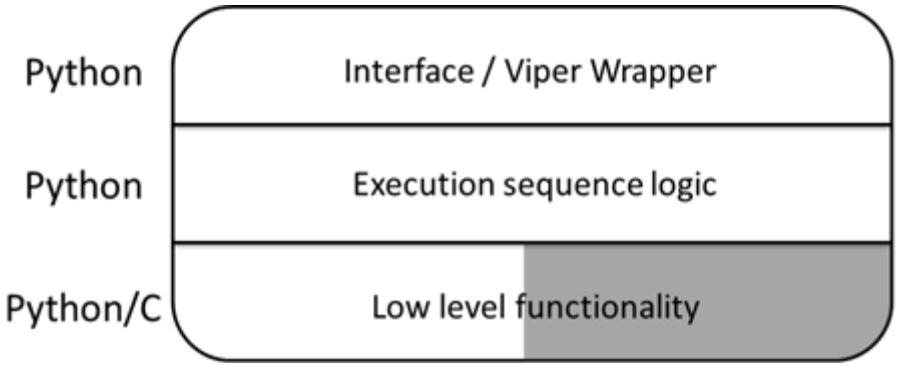

Figure 12 - Layers in GEF

This approach promotes the modularization of the implementation. Additionally the layer above does not have implementation knowledge of the layer below and only knows the provided functionality. This makes it easier to make changes without affecting the surrounding parts. Layering the component helps to support separation of concerns that bring flexibility and maintainability.

8.2.3. Python and C Combination

Python is gaining ground in ASML. It has been used more and more in parts of the software. This is possible because within ASML there has been developed a domain specific interface language. This interface DSL is used for communication between components as well as between parts of the same component. The main characteristic of this interface DSL is that it allows the two sides of the interface to communicate without having the same implementation language. This allows the use of multiple languages and makes much easier to introduce a new language in the future.

The new design of GEF is taking advantage of this flexibility to utilize both languages. In most cases Python is used. The top layers of GEF are using Python to take advantage of object oriented features and promote a clear and modular structure. The general approach is to use Python whenever possible and avoid the long and hard to understand C equivalents.

In some cases, use of C is inevitable. ASML software is moving towards multi-language support through the use of the ASML interface DSL. The majority of components support this DSL interface. However this process is not universally adopted and there are still cases in which components can have only C clients. Additionally, taking into consideration the effort of re-implementing the entire GEF in Python, it makes sense to re-use existing implementation parts to speed up development and reduce required implementation time.

For the above reasons, the new GEF design is a hybrid Python-C design that tries to use each language when the situation calls for it.

8.2.4. Viper Model View Controller Pattern

An important requirement identified is the separation of UI and business logic in GEF. Currently, GEF implements the UI that the clients are using. It provides them the functionality to create the graphical interface layout. This is highly coupled with the business logic of GEF. The details about the UI implementation can be found in multiple parts of the existing execution sequence. It was pointed out that this is one of the significant issues that the new design must tackle. This separation would make GEF more modular with changeable parts and fewer dependencies.

Since Viper is the provider of the UI functionality in the new design, it is obvious that it has an important role in solving this problem. In fact, Viper has a design based on the Model-View-Controller pattern as shown in Figure 13.

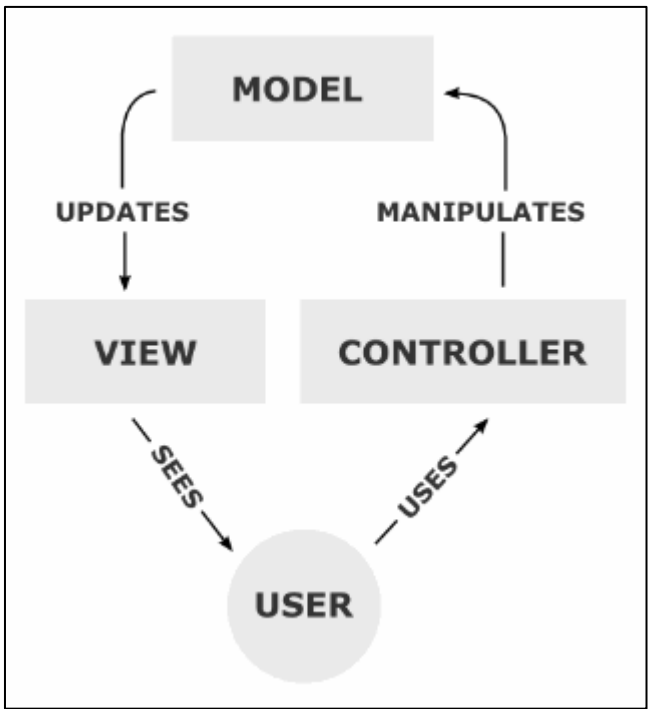

Figure 13 - Model-View-Controller pattern

Model

The Model in Viper contains the application data. The data are defined by the creator of the CPD test individually. Viper itself does not contain any data but allows its clients to set them. In the case of the GEF design, the application data used by Viper is a combination of CPD-specific data from the client and common data from GEF. The data that exist in the Model part are persistent throughout the test and can be used in all the phases of the execution.

View

The View is defined in an xml format file with the use of a UI builder tool. The xml file is highly dependent on the data model and vice versa. Widgets are linked in the xml file with corresponding data models. If a value of a widget changes, then the value of the data model gets updated also. Viper transparently provides this observer pattern functionality.

The xml file-based view is built separately by each test. The GEF common part is always present for GEF clients (xml template) and the client can modify it to add a client specific part.

Controller

The Controller in Viper is responsible for executing the business logic of the particular CPD test. The execution steps are defined by the clients and passed to Viper. The Controller exe-

cutes these steps in the defined order. Data models are available to the Controller and are updated during the execution. As a result the widgets that are linked to data models are notified for any changes. In the case of GEF clients, the common GEF execution steps are combined with client steps and then passed to Viper's Controller.

To summarize, Viper already offers the separation of the UI, the business logic and the data models. GEF benefits from it. The contents of the Model, View and Controller are defined in GEF for the common part and by the GEF client for the client-specific part. The way Viper expects tests to be organized dictates that GEF also will follow the same loosely coupled way.

8.3 Interface

The interface of GEF towards the clients was the main source of issues in the existing implementation. The new design must have an interface that provides the common GEF functionality to its clients and at the same time allows the extension of that functionality by its clients.

The interface of the new design exposes the following:

- The Viper functionality.
- The data models that are needed by the client for client-specific extensions.

8.3.1. Viper Wrapper

The idea behind using Viper and implementing GEF as a framework is to intercept the Viper functions that normally a test uses from Viper directly. This way the sequence can be controlled by GEF and the common functionality can be added.
The client uses the GEF interface the same way it would use the Viper interface. As long as it conforms to the Viper way of structuring the test, GEF will handle the communication with Viper.

8.3.2. Callbacks and Client-injected Functionality

The existing way of supporting test-specific functionality alongside the GEF common execution sequence is the use of callback functions. These are implemented by the client and called in specific points during the execution.

The data that callbacks can use are the data that GEF is exposing in its interface. These are the commonly used symbols that all tests are aware of. Additionally, the client can define client-specific data that can also be used by the callback functions. This way, when a callback function is called in a specific point of the GEF sequence, these client-specific data can be used to extend parts of GEF such as the Test Log, the Test Report, without the need of GEF to be aware of them. This way GEF remains clean from client-specific content.

The new design maintains the callback mechanism in order to remain compatible with the execution sequence logic and also understandable to developers that were familiar with the callback functionality. Clients can still implement functions that are going to be called in specific points and perform the additional client functionality.

In order to provide an additional way for functionality injection, the new design introduces two new concepts:

- **Step injection:** The execution sequence of GEF is decomposed into steps. The client can implement additional steps and inject them between the common GEF steps. This is suitable for small scale functionality alternations.
- **Step implementation selection:** The common steps of GEF can have multiple versions in order to cover different client groups. Thus selection of which implementation is needed is possible for the client. This is suitable for large scale functionality alternations without affecting the existing clients.

8.3.3. Fine-grained Interfaces

The most significant problem of the current GEF is the existence of one large interface that is used by all clients. This interface exposes not only the common GEF functionality, but also functionality that was added for specific test needs. Therefore tests have knowledge of information that does not concern them. This has led to the major issue of client impact whenever the interface requires a change. This issue is not related to the implementation language or the CPD Framework used, but it is a consequence of poor evolution of the interface itself.

The solution to this is to break the interface into smaller ones based on the Interface Segregation Principle. One interface would provide the most common and stable functionality of GEF that all clients will need. Then, there will be smaller interfaces that will group relevant functionality. Clients will only use the interfaces that they require, thus breaking the unneeded dependencies. The two interface structures can be seen in Figure 14 and Figure 15.

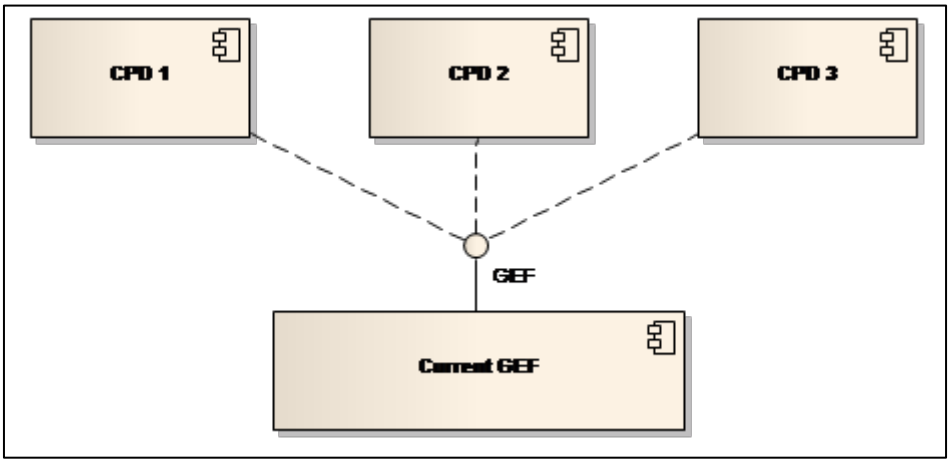

Figure 14 - GEF current interface structure

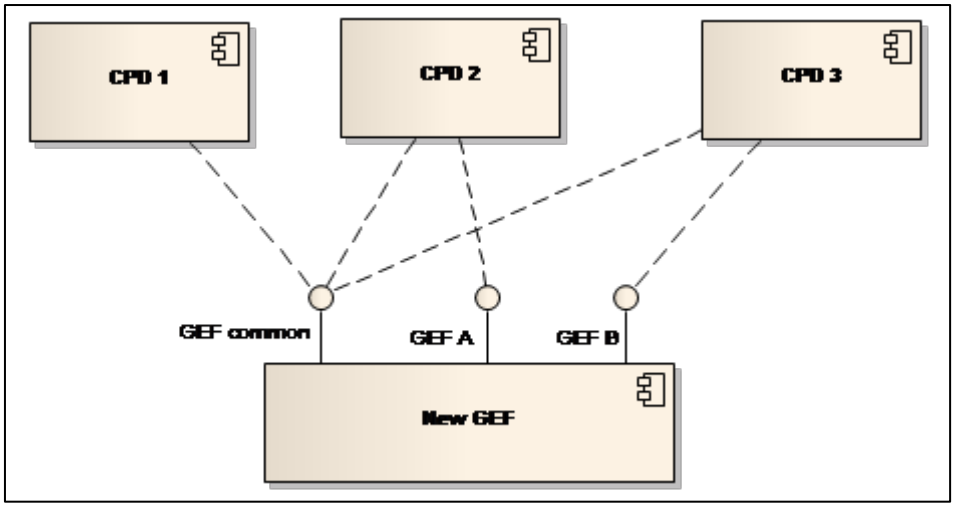

Figure 15 – GEF new interface structure

The result will be decreased impact when changes occur to these interfaces due to a smaller number of clients. Now the chances of changing the common GEF interface are decreased by spreading functionality to the other interfaces and applying modifications only where needed. ■

9.System Design

Abstract – After explaining the main architecture decisions, this chapter provides a more detailed look at the GEF internal design. In Section 9.1 the high level view of the design is given. In addition the layers of the new framework and their responsibility are explained. The client part is explained in Section 9.2. In Section 9.3 the sequence of the execution can be better understood. Furthermore, the sequence diagram there makes the interaction of GEF components more clear.

9.1 Internal Design

The layered approach that was described in the previous chapter is shown here in more detail. The layers of the final design are the

- Interface
- Application
- Execution Logic
- Low Level Functionality

Another entity that exists in the diagram is the Data Models. This is placed vertically to show the usage of these domain models throughout the layers. Figure 16 shows the GEF design from the package point of view.

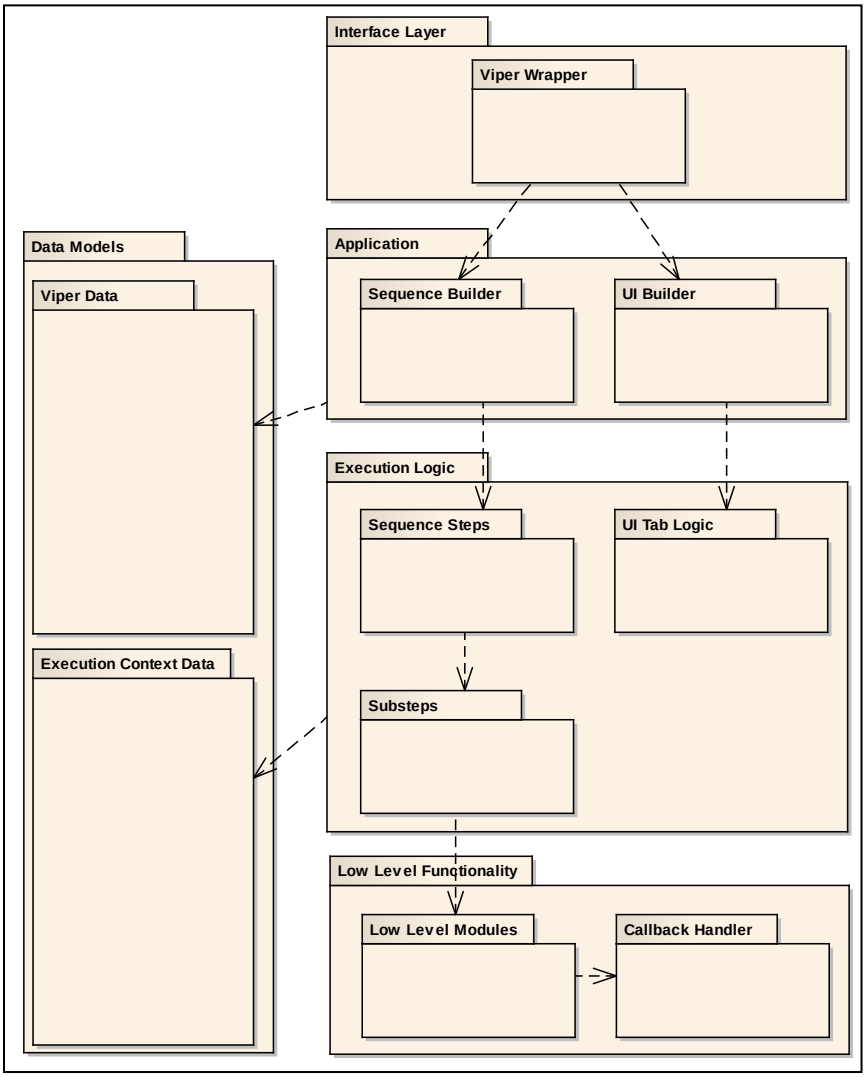

Figure 16 - Overall GEF architecture

The layers of the GEF design are explained in the following sections.

9.1.1. Interface Layer

GEF is a framework that will provide functionality to an increasing number of CPD tests. Based on the evolution of the current GEF throughout the years, it can be safely assumed that the number of clients will steadily increase for the new GEF also. This characteristic was taken into consideration when the interface structure was designed.

The new interface must support the addition of new functionality when it is required by its clients. At the same time the existing clients must be affected as little as possible. This ability has proved to be the biggest challenge, because it implies the knowledge and anticipation of possible future changes and needs of GEF's clients but also changes in external components that GEF depends on.

The solution for an extendable interface was based mainly on splitting the GEF interface into smaller ones based on functionality grouping and commonality of use between clients. This has been tried out in the prototype of the new GEF.

Additionally, the GEF interface is in essence a wrapper around the functionality of Viper and for that reason the structure for the provided functionality is relatively the same.

The interface of GEF exposes

- GEF functions
- GEF data types

GEF Functions

The Viper interface exposes to clients the functions that allow the tests to perform certain actions. The most important ones include the configuration of the UI, the configuration of the execution steps and the start of the test. GEF is responsible for adding to the test-created content, the additional common content that is needed by all tests. So in the case of the UI, the test creates the UI elements and behavior that are related to the test-specific needs. The rest of the UI layout is provided by GEF.

In order to allow this combination of test-specific and common content, GEF merges it in a way that is transparent to the test. This behavior is achieved by the Viper Wrapper component. The main idea of this part can be depicted in Figure 17.

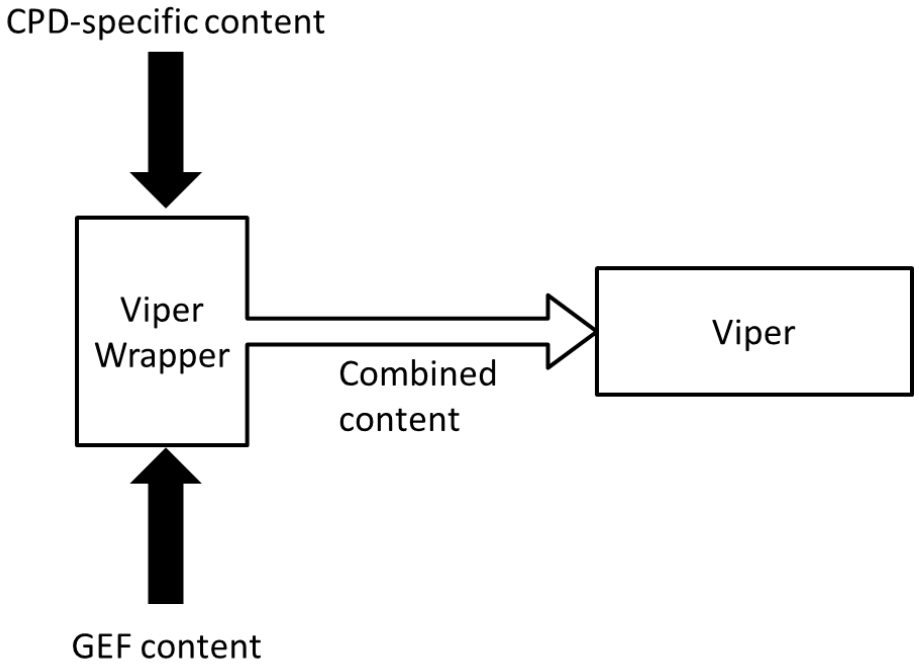

Figure 17 - Viper Wrapper component

For each function exposed by Viper, GEF offers an equivalent version in its own interface. The clients of GEF can call the same functions and expect the same functionality. GEF in a client-unaware way injects the additional content before finally delivering it to Viper. An efficient and elegant way to implement this was with Python decorators.

GEF Data Types

GEF clients can add test-specific functionality that is executed in certain points of the common sequence through the callback functions. This means that clients need to be aware of particular data that are used in GEF. The grouping of these data plays an important role in the extensibility of the GEF interface. As seen in the current GEF implementation, the one large data interface that GEF exposes to its clients creates significant problems whenever a change is required. The solution was given by the creation of new smaller interfaces for non-common and test-specific data. This way the changes are not always occurring in the same interface but are distributed. The Interface Segregation Principle is the followed approach.

The first version of the new GEF data interface is considered common and used by all clients. In the future, when new requirements might appear and new data need to be exposed to a new client, then the place of the new data must be considered. If the new data add value to the existing clients then the common data interface will be updated. Otherwise, if existing clients

have no benefit from the new data, then the creation of a new smaller data interface will be the way to go.

9.1.2. Application Layer

The Application layer is responsible for the combination and preparation of the content that comes from the client and from GEF. It is called by the Viper Wrapper with the test data as input and provides the combined data from the test and GEF as output.

The combination logic resides in Builder components that perform the merging and return the result. Viper expects a specific structure in order to execute the test sequence and create the test UI. This is being composed in the

- Sequence Builder
- UI Builder

Sequence Builder

The Sequence Builder receives the parameters from the test and creates accordingly the execution structure. This is done with the use of a step factory design pattern that is creating the appropriate step instances according to the test's requests. This way the test can decide which step implementations it requires. The presence of the factory pattern liberates the client from knowing the concrete step implementations and creates a layer of abstraction. Therefore the creation of additional implementations of steps in the future is transparent to the clients and makes GEF more extendable. The implementation details of the steps are defined one layer lower, in the Execution Sequence Steps component.

The created GEF sequence is then combined with the additional steps that the client might have defined. This combination of steps is converted to the form that Viper is expecting. Finally it is returned to the Viper Wrapper which forwards it to Viper.

UI Builder

The UI builder receives the UI related structures defined by the test, for the test-specific UI layout and details. It then prepares the UI structures for the common GEF UI part. Finally it combines both test-specific and common content and returns it to the Viper Wrapper. Eventually Viper receives the complete structure of the UI.

The UI is composed of tabs. GEF creates the tabs that are common between all CPD tests. These tabs are considered stable and changes are not frequent. The reason is that operators are used when working with a particular UI and having differences between CPD tests will create problems and difficulties during operation of the machine. Apart from the common GEF tabs, additional tabs can be added from the client. These tabs are not expected to be complicated. The few additions that the CPD requires are gathered in these tabs to avoid changes in the common UI.

The behavior of the UI elements is also defined by the UI Builder. Widgets perform particular actions and the data models are being changed accordingly. These actions are implemented one layer lower, in the UI Tab Logic.

9.1.3. Execution Logic Layer

The Execution Logic Layer has the responsibility of defining the

- Execution Sequence Steps
- UI Tab Logic

Execution Sequence Steps

The concrete steps that are executed throughout the CPD test are implemented here. These steps are composed from substeps which lead us further down, towards more low-level and fine-grained functionality.

The creation of the steps was done by interpreting the original sequence and finding the points at which the sequence could be split. The created steps consist of related functionality. The five steps that depict the division of the sequence are

1. **Initialization**: The required actions that are needed to set the machine in the correct state before exposing is performed. Additionally, data models are initialized and the required memory space is allocated.
2. **Exposure:** The low level exposure functionality is invoked after the necessary parameters and inputs have been set. During this step, results and notifications from the exposure component are being used to fill data models that are needed for the Test Log and the Test Report. At the end of this step the exposed wafers have been processed and finalized.
3. **Test Log Creation:** The data collected during the exposure are processed and stored in a Test Log. The Test Log is saved in a file so that it can be used later by other components.
4. **Test Report Creation:** The data collected during the exposure that are needed for the Test Report are being processed and placed in the right structures to form the Test Report.
5. **Finalization:** The allocated memory is freed and the machine is brought back to an idle state.

With modularity in mind, these five steps are composed out of substeps. Substeps are defined in the low level functionality layer or on the data model layer. The substeps help to keep the sequence fine-grained and help to minimize the impact of changes in GEF. For example the Exposure step is composed of the Exposure Initialization, Start Exposure, Perform Exposure Verification, Store Exposure Results substeps.

The sequence of the current GEF is implemented in a procedural way. In order to keep the functionality the same, this sequence had to be mapped to the new design. This mapping can be identified in the order of steps and substeps in the new design. The difference is that the actions are performed by classes that are responsible for particular objects. Therefore implementation details have been taken away from the sequence and have been moved to separate classes.

UI Tab Logic

As already mentioned, GEF is responsible for creating the layout for the common UI tabs that exist in all tests. The widgets and structure of these tabs along with their behavior needs to be provided to Viper by GEF. Then Viper will be able to create it when the test is executed.

In Section 9.1.2 the UI Builder was explained. The UI Builder receives the test-specific UI from the test and the common GEF part from GEF and then combines them and forwards them to Viper. The common GEF part is created in this level in the UI Tab Logic part.

The UI Tab Logic mainly defines the extra behavior of GEF widgets. Some widgets like text labels that are directly linked to a variable do not require any additional implementation. Viper will take care of updating the values in the data models that correspond to that widget. However, in the case of buttons and other more complicated widgets, specific behavior is expected. For example, when the exposure recipe file is selected by the user, GEF must perform checks that ensure that the particular recipe can be used on this occasion and update more than one field. On other occasions, selections in the UI can lead to hiding of UI elements, change of options in another widget, etc. This kind of behavior is implemented in the UI Tab Logic.

9.1.4. Low Level Functionality View

This layer contains the core implementation of the functionality used in the execution steps. Additionally, the Callback Handler module that offers the execution of the test-defined callback functions is located here.

Low Level Modules

This part is responsible for the execution of calculations and usage of infrastructure components that are needed during the test execution. The main characteristic of this part is that it is

implemented in both Python and C. This adds some complexity since the communication between language and C has to be handled.

The C part has as its main purpose to host functionality that is needed in GEF but it cannot be used in Python. This means that the components that provide this functionality have not yet converted their interfaces to the mandatory ASML interface DSL. The only way to use this functionality from Python is to have an intermediate C layer. Communication between C and Python is done through the ASML interface DSL and then the C part can execute the necessary calls to the external components. In the ideal situation the C part would not be needed and all calls would be possible through Python. Until that point is reached, this walk-around must be used.

Apart from incompatibility of other components, the C part provides one more service. It allows the re-use of some parts of the original GEF. In some cases it is possible and efficient to use a part of the existing implementation instead of migrating it completely in Python. By using the ASML interface DSL, Python can call upon such parts and use them in the execution sequence. The usage of existing parts is not always possible and sometimes the effort to re-implement the same part in Python is more efficient.

The Python part provides functionality that steps and substeps are requiring. This is either calculations that set data models or calls to functionality from other components. This part contains mostly independent functions that work on input provided by the Execution Logic Layer. In general the Low Level Functionality Layer has knowledge of the infrastructure that GEF is using and hides this knowledge from the levels above.

Callback Handler

The Callback Handler module provides the callback functionality that was described in Section 8.3.2. At the beginning of the test, the client registers through the GEF interface the callback functions that it has implemented and wishes to use during the execution. These callback functions then reside in the Callback Handler module. This module allows the registered functions to be called by the rest of the GEF components. The specific points that these functions are called are in the Low Level Modules. During execution, the appropriate callback function is called and the test-specific functionality is executed.

A characteristic of the Callback Handler is that the interface that it exposes for the invocation of the registered callback functions uses the ASML interface DSL approach. The reason for this decision was the need to call the callback functions from both the C and the Python part of the Low Level Modules. As already mentioned, the core functionality where the callback functions are mapped is implemented partly in C and Python. This means that the Callback Handler had to be callable from both parts.

9.1.5. Data Models

The Data Models contain all the domain data that is used throughout the execution of a CPD test. The Data Models can be split into two main parts

- Viper Data
- Execution Context Data

Viper Data

The Viper Data represent the instance of the data that are being used by Viper during the execution of the test. In the Viper way of creating a CPD test, a set of data must be defined by the client. This must contain whatever the client requires during the test and also the data that are needed for the composition of the UI. This is the case also with GEF. This data set is defined in the client and it contains the combination of test-specific data and GEF common data exposed by the GEF interface.

The Viper Data are an instance of the defined data set. The same instance is used from the beginning to the end of the execution. It behaves as a singleton and can be manipulated by all modules used in the execution. This requires some additional attention to avoid unwanted simultaneous access and manipulation of the Viper Data instance.

The important difference between the Viper Data and the rest of the Data Models is that the Viper Data are available to the client also. The client can use the Viper Data in its callback functions and additional steps. Also the Viper Data are used by most of the layers of GEF in contrast with the rest of the Data Models that are mainly used by the Execution Logic and the Low Level Functionality Layer.

Execution Context Data

The Execution Context Data contain data that are used throughout the execution sequence but only by GEF. The client is unaware of these data. Some of the data follow the Viper Data singleton approach and are accessible by multiple parts of GEF.

The Execution Context Data contain also classes that are responsible for initialization, manipulation and finalization of specific data entities. Thus responsibilities for particular data types have been grouped into classes to encapsulate and hide the details from the parts that use them.

9.2 GEF Client

The GEF client belongs to the category of CPD tests. It is different from other CPD tests because it requires wafer exposure. The GEF client uses the interface provided by GEF. This interface as already mentioned, is composed of two parts, GEF functions and GEF datatypes. The GEF functions are based on the Viper functions and allow the client to define specific aspects of the test. The GEF datatypes are provided in case the client wishes to have additional functionality during execution and must work on data that exist in GEF.

The GEF client is based on the Viper way of creating CPD tests. This makes the creation of the GEF client very familiar to developers. The client defines

- **Additional steps**. The client can implement more steps that will be executed alongside the GEF common steps.
- **The complete UI XML file**. This file is based on a template from GEF that already contains the GEF layout. The client has the option to add content to the XML file for test-specific tabs. This way the common GEF UI remains untouched.
- **The test-specific UI tab behavior**. The client can define the behavior of the additional tabs and widgets that it has introduced in the XML file.
- **The callback functions**. The client is needed to specify if the use of callback functions during execution of the test is required. These callback functions must be implemented by the client and then registered to GEF. The callback function definition is the most significant difference from a Viper CPD test and a GEF test.

The GEF clients differ from the rest of the CPD Viper clients in that they implement less logic. The common part already exists in GEF and the client provides mostly parameters and some additional functionality.

9.3 Test Execution Process

The execution of a test involves the use of numerous external components. Many of these dependencies are inherited from the existing design and are needed to fulfill the functional requirements of GEF. There are some that were introduced due to the new design. The most important of those is Viper. These modifications of the design change significantly the interactions between objects and components. The sequence diagram in Figure 18, attempts to depict how parts interact with one another and in what order during a CPD test execution.

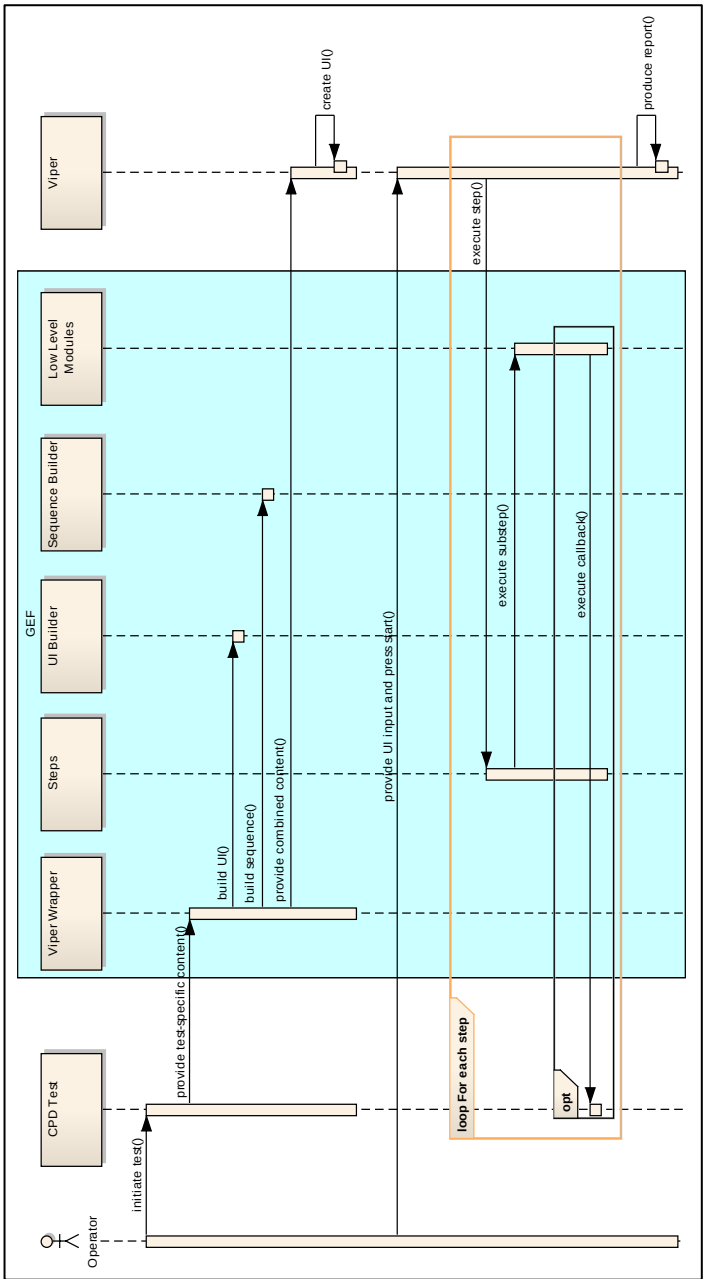

Figure 18 - GEF execution sequence diagram

Although details have been removed and the sequence is shown in a very simplified way, the order of execution and the part interaction can be seen. The mediating role of GEF between the CPD test and Viper is clear by seeing that every call goes through GEF and it is up to GEF to decide how to handle it. During execution, the steps provide most of the functionality, and occasionally the test provides some additional functionality through the callback functions. It can also be seen from this diagram how the previously owned responsibilities of GEF, such as UI creation, execution control and report creation have been moved to Viper.■

10. Implementation

Abstract – In this chapter the implementation of the design is described. The role of the implementation was mainly to prove the feasibility of the design and to discover pitfalls that would require additional effort. Feasibility implementations were used for decisions in the early stages and prototypes for the later stages of the project.

10.1 Approach

The implementation of the solution was done based on the exploratory nature of the project. This mainly meant

- Feasibility Implementations
- Prototypes

10.1.1. Feasibility Implementations

The feasibility implementations focused on specific parts of the design. This was mainly done in the first part of the project, when the design and architecture was created. In the early stages of the project, there were many vague areas that had to do with applicability of specific ideas in the ASML software domain. This meant that for certain decisions to be taken, small proof-of-concept prototypes had to be built.

In the cases of isolated implementations that did not require being part of the ASML software, Eclipse IDE was used. There, ideas that had to do mostly with the use of Python features were tested. This approach was very convenient and had minimum overhead.

For feasibility implementations that had to be part of the ASML system software another ASML facility was used. It is possible to have a development view of the system software that serves exactly the purpose of proof of concepts. It is not possible to integrate this view with the rest of the software, so it is perfectly safe to try ideas and examine their impact. More complicated feasibility implementations were developed this way. This approach provided significant insight before important design decisions were taken. It also helped to increase the understanding of the author for the domain specific details by allowing a risk-free hands-on experience.

The feasibility implementations greatly contributed in the first phase of the project. The design that was created was based on findings in these implementations.

10.1.2. Prototypes

After the creation of the architecture and design of the new GEF, there were still areas that had not been tested. Feasibility implementations covered some fundamental issues concerning applicability of decisions. However, many unanswered details concerning larger scale implementation remained. All these details had to be explored. A series of prototypes was planned for this purpose.

The prototypes implement features that are essential for the new GEF design. Through these prototypes, stakeholders can be convinced about the feasibility of this new approach. The new design introduces major changes and this creates a sense of insecurity. This feeling can be minimized through prototypes that showcase the new design running in the actual system software.

Additionally, the prototypes provide insight into how the work break down structure of the implementation of the new GEF can be planned. Each item there can be better estimated since the grey areas will have already been tried out in the prototypes.

The prototypes were implemented in versions. The features of the new GEF whose applicability needed to be tested were put on a list and then distributed amongst the prototype versions based on their importance. At the end, each prototype implemented additional features to the

previous version until the final version offered all the required features. During the prototype creation phase, observations and conclusions were documented. This ensures that they can be used later on to plan the implementation of the new GEF accurately. The final prototype version served as a mean for demonstrating the new GEF design. The Prototype Design Document [10] explains the details of the prototype. ■

11. Verification & Validation

Abstract – In this chapter the capability of the new design to provide the expected functionality is shown. The two most important outputs of GEF are compared between the original version and the prototype. The comparison shows that the prototype can successfully create the expected output and fulfill its role in the calibration sequence. Also the realization of the requirements of Chapter 6 by the prototype is shown.

11.1 Output Comparison

The feasibility nature of the project made the comparison of the two frameworks as black boxes the most important means to verify the correctness and fulfillment of requirements for the new design. It might be the case that the prototype of the new design is lacking some features that should be there in the final implementation. However, it successfully serves the purpose of proving that the new design is capable of achieving the expected outputs.

As mentioned earlier, the two outputs of a CPD test that uses GEF are the

- Exposed Wafer
- Test Log file

In order to verify that the new design is producing the same results as the old, these two outputs had to be compared. The same CPD test was used in both the current GEF implementation and the new GEF prototype. This means that the inputs were identical in both cases. Also both tests were performed on the same system with the same parameters and specifications. The test chosen is one that utilizes most of the functionality of GEF and represents the majority of GEF CPD tests.

The results are compared separately for each output. The comparison was performed by using a file comparison tool called kdiff3. This tool is used for identifying differences and merging files in the ASML UNIX development environment. Comparing the Test Log files was done in two ways:

- Comparing the existence of all the data structures
- Comparing the values in the data structures

11.1.1. Exposed Wafer

The exposure of a wafer is carried out by the Lot Production component. GEF needs to provide Lot Production the necessary input with the parameters for the exposure. In the new design providing the appropriate parameters was not the part that presented a challenge. The fact that the new design is based on Python introduced a layer of uncertainty in the feasibility and applicability aspect. This included the implementation of the exposure part and the ability to use the Lot Production component.

The Lot Production usage had to be identical to the original GEF and had to return the same results after the wafer exposure was finished. Lot Production returns a data structure with the results of the exposure.

The use of Lot Production was achieved through Python in addition to all the secondary functionality required to support Lot Production. Continuous try outs of the Lot Production were performed in order to identify possible unwanted behavior. The implementation eventually reached a level of maturity that can be considered stable and reliable to use. This means that the concern about exposing a wafer in Python has been eliminated.

Additionally, the data structure that Lot Production returns after exposure was compared between the current and the new design implementation. As mentioned a specific CPD test was used to provide the input to both designs. The results of the two structures that were returned by Lot Production were 100% identical structurally and the only differences were the date

field values. This proved that the exposure achieved the same results from Python as it did in the original C implementation.

11.1.2. Test Log file

The Test Log file contains all the information related to the exposed wafers. It contains the input that the exposure was based on and it also contains results of the exposure and additional information that were gathered during the execution of the test. The Test Log file is in essence a combination of multiple data structures that are glued together throughout the execution of a CPD test by GEF. In order to make sure that the new design is creating a Test Log file that contains the same data structures as the old, the existence of these data structures had to be checked.

The creation of the Test Log file is a complicated process. GEF is responsible for collecting all the data, performing specific calculations on them and eventually stitching them all together in one large file. The average file size of a Test Log after the exposure of just one wafer is 500,000 lines. The creation of a Test Log file identical to the original required converting large part of the C implementation to Python. Since the line-by-line comparison was the measurement method, there could be no parts missing from the Test Log creation of the new design prototype.

The two Test Log files were compared first regarding the structural similarity. After incremental additions of Test Log creation logic, the resulting file from the prototype was identical to the file from the original GEF. Essentially all the data structures that were expected in the file were present and no parts were missing.

Regarding the values of the fields in the data structures, a comparison was done. In this case differences were found in the values. This occurs in a small number of values compared to the size of the files. In order to find out if the differences in these values are occurring because of the new Python implementation of the prototype, differences between consecutive executions of the same CPD test with the original GEF were also checked. Table 6 shows an example of results that had differences in the Test Log. The first two columns show the value after running the same CPD test in the original GEF and the third column has the value as found in the Test Log from the prototype executing the same test.

Table 6 - Sample values from the Test Log comparison

Original GEF Test run 1	Original GEF Test run 2	Prototype GEF
32.0095572309**6**1e-21	32.0095572309**60**6e-21	32.0095572309**60**4e-21
222.52019649**4**535e-24	222.52019649**6**73e-24	222.52019649**4**598e-24

Since the differences are small and also existent between Test Logs from the original implementation then they are considered normal fluctuation based on slight changes in the state of the system. This proved that the new version can achieve identical functionality as the original one and provide the very crucial Test Log files.

11.2 Change Impact Comparison

As already mentioned, one of the major issues of the current GEF is the large amount of effort that is required for every change that it has to go through. This includes interface, UI and internal changes. Several change examples have been formulated and their impact on the two designs is compared. These tables due to sensitive information can be found in the internal ASML document with id number D000214650 and title Change Impact Comparison.

The conclusion of these comparisons show a significant improvement in the impact and effort required when changes have to be applied to the new GEF design.

11.3 Requirements Revisited

In Chapter 6, the desired requirements of the new design were listed. At this point, the fulfillment of those requirements in the produced prototype is analyzed.

11.3.1. Functional Requirements

FR1.1: GEF provides a UI mechanism to the client tests.
Fully Implemented. The prototype as mentioned in the design of the new GEF provides through Viper the UI creation mechanism that is commonly used for ASML user interfaces.

FR1.2: GEF provides the default/common part of the UI.
Partially implemented. Since the common parts have not yet been decided, the prototype provides a simplified layout in the place that the common UI will be.

FR1.3: GEF has a logical division of UI into tabs with widgets grouped based on functionality.
Partially implemented. The division of the UI is made into common and test-specific. One tab is made for the test-specific widgets and two tabs for the simplified common tabs. More common tabs are expected in the final form.

FR1.4: GEF uses similar parts of the wafer production screen UI.
Fully implemented. The look and feel used by Viper's UI mechanism is identical to the one that is used in the production screen.

FR2.1: GEF provides a common stable interface for test clients.
Not implemented. The prototype uses the current interface of GEF as a starting point. The investigation on which symbols should be in the interface has to be done before the final common interface can be defined.

FR2.2: GEF allows the extension of the interface for test clients with varying requirements.
Partially implemented. The prototype shows that additional interfaces can be implemented to host the client-specific symbols needed. No particular case was tried since this option does not introduce a significant risk regarding feasibility.

FR3.1: GEF provides hooks to test-specific functions during execution.
Fully implemented. The support of the callback function mechanism is fully implemented by the prototype. The hook points of the new GEF are the same as of those of the existing GEF. Clients have the option to implement the same functionality as they would in the current GEF.

FR3.2: GEF supports input from the UI.
Partially implemented. The prototype provides an input UI that parameters can be set. The input UI is very much simplified though in the prototype and the final input UI will contain much more fields.

FR3.3: GEF supports input from the test clients.
Fully implemented. The prototype supports additional input from the client that is then combined with the common GEF content. This client input is related to the UI layout, additional sequence steps and client-defined callback functions.

FR4.1: GEF is able to create and store Test Logs.
Fully implemented. The prototype follows the original sequence for the creation of the Test Log files. This includes collection of exposure data, invocation of external components that create, populate and store the Test Log and the hook point to test-specific Test Log actions through the invocation of a callback function from the test.

FR4.2: GEF is able to add client specific part in the Test Log.
Fully implemented. The prototype provides the hook for the necessary client-defined callback function during the creation of the Test Log. At that point the client can apply the desired

actions to the Test Log. In the prototype additional content was added to the Test Log through the callback function.

11.3.2. Non-functional Requirements

NFR1.1: GEF is a framework for providing wafer exposure functionality to calibration, performance and diagnostic tests.
This general requirement is satisfied by the prototype by providing an interface for clients and functionality that allows the creation of calibration tests that utilize this functionality.

NFR1.2: GEF uses the wafer production engine, so calibration sequence and production act alike.
The GEF prototype is successfully using the same component and functionality for exposing wafers as in the normal production sequence.

NFR2.1: GEF provides a UI with the same look and feel as the rest of the calibration software.
The use of the Viper-provided UI engine ensures that the clients of GEF are conforming to the dictated UI look and feel. The created UI for the prototype is simple in regard to content but it follows the appropriate layout and coloring rules.

NFR2.2: New GEF design provides a more convenient and flexible way of creating CPD tests.
Convenience is ensured by the use of Viper. The Viper way of creating tests is familiar to developers and it provides points for extension. The prototype client is made the same way a normal Viper client would except for some GEF specific additions.

NFR2.3: GEF extensions (in functionality and in UI) for new clients have small impact on existing clients.
The prototype supports the creation of client-specific steps and these are injected between the common sequence without the knowledge of GEF or the rest of the clients. Also client-specific UI needs are put into the dedicated client tab and do not affect the rest of the common UI, thus neither GEF nor other clients.

NFR3.1: Manual for creation of new exposure tests based on the new design.
An initial guide covering the creation of clients has been created in the appropriate EMEX ASML wiki page [11].

NFR3.2: GEF's implementation is understandable.
The prototype is much smaller than what the final GEF will be. But it can already be seen from the prototype that the migration of particular parts to Python has resulted into much smaller and readable implementations. The average size reduction of similar code to Python is a factor of three. This is a strong step towards increasing the understandability of the component.

NFR4.1: GEF allows client CPD tests to extend the common functionality.
The new GEF design has a mechanism to allow extension of the common functionality without creating a large impact on the current clients. The clients can select the implementation version of each step of the common sequence. This was tried in a small degree by the prototype. However more investigation is needed to establish the most efficient way of having different versions of steps in GEF.

NFR4.2: GEF should not have test-specific knowledge but shall be kept generic and unaware of test-specific details.
The prototype was based upon the current form and contents of GEF. Thus, additional investigation must be done in order to decide what is test-specific and what is common in the current contents.

NFR4.3: GEF must have a decoupled UI from business logic.

This was satisfied in the prototype by the use of Viper's observer pattern. Value changes are done to the datamodels linked to the widgets and UI values are updated. So there is no UI technology-specific knowledge in the logic of the prototype.

NFR5.1: GEF fits the current software architecture and concepts used in ASML.
The prototype is following the ASML specified way of working and is developed on the latest software baseline of the lithoscanner system. Also it is based on Viper, the latest CPD framework according to ASML guidelines.

NFR5.2: GEF's new design is used for new tests (no backwards compatibility).
This is indeed the case with the prototype since the interface towards the new clients is adjusted to Viper and cannot be used for the current clients.

NFR5.3: GEF is compatible with the rest of the calibration sequence that sets system parameters based on exposure on a wafer and readout of the result.
The prototype functionally focused on creating the same output as the current GEF. Section 11.1 showed how this was achieved. Since this output is the one used by the next steps, it makes the prototype compatible with the calibration sequence.

11.4 Conclusion

The important deliverables that are expected from GEF by the rest of the calibration sequence were successfully created. Both exposed wafer and Test Log file were similar to an acceptable level to the original output and that proves that the new design is fully capable of providing the GEF functionality. Additionally, the requirements gathered in Chapter 6 are mostly satisfied by the prototype and proven to be feasible, clearing the way for the final full scale implementation of the new GEF.■

12. Conclusions

Abstract – In this chapter the results of this project are shown. The two main achievements are the new design proposed for GEF and the prototype that proved its feasibility. Furthermore, the future steps that ASML can follow as a consequence of this project are mentioned.

12.1 Results

The goal of the project was the creation of a new software design for the GEF component that will deal with the problems of the current one. It should make changes on the framework to require less effort and have less impact on the clients. It should provide additional UI functionality to cope with client-specific needs. Also the implementation itself should be more maintainable and readable to ensure increased life expectancy. This new design should be implemented in a prototype to prove its feasibility and showcase the new features. The aforementioned goals are considered as achieved.

12.1.1. New Generic Exposure Framework Design

A new design for the Generic Exposure Framework has been created. This new GEF design focuses on solving the problems of the current component. As shown, it provides new features that tackle the current issues and bring the framework in-line and up-to-date with the ASML-specific way of working. It also satisfies the quality characteristics that were defined in Chapter 6.

The beneficial characteristics are

- **Customizable UI mechanism**. Client and common UI are now separate. This makes the common UI stable and reusable. Additionally the client has more freedom to create its own client-specific UI. This satisfies the flexibility requirements that were set at the start of the project concerning the UI mechanism.
- **Flexible Interface towards clients with reduced change impact**. The grouping of exposed interface symbols based on the role and functionality leads to smaller fine-grained interfaces. This reduces the impact of possible future GEF interface changes. It also promotes encapsulation since clients use only the interfaces that concern them and are unaware of unneeded symbols.
- **Smaller, more understandable and maintainable component**. Parts of the current GEF are obsolete since Viper is responsible for providing the particular functionality in the new design. Furthermore, because of Python the implementation will be smaller, more modular and self-documenting. The result will be a more readable and maintainable component.
- **Reduced implementation effort**. Because of Viper's offered functionality the implementation effort of the new design is significantly lower. It is a fact that Python speeds up implementation compared to C. From this, both GEF and its clients will benefit.
- **Compliance with ASML software standards**. Viper has become the standard ASML-specific way of working when it comes to CPD tests. The use of Viper as a base for the new GEF design brings the component and the clients using it in-line with the rest of the ASML software architecture. Moreover, the UI offered by Viper is compliant to the standard UI look and feel that CPD tests must have.

12.1.2. Prototype Implementation

The proposed design's feasibility was proven by implementing it on a prototype. This prototype is the first step towards the new version of the framework. Through the prototype the new characteristics of GEF were showcased. Moreover, the important functionality was tried out and validated as seen in Chapter 11.

The prototype assisted in

- **Proving the new design's feasibility**. As a proof of concept, the prototype ensured that the proposed solution is possible in the ASML system. The success of this pro-

ject could not be guaranteed by the creation of an improved design unless this was tried and effectively executed in the ASML software environment.

- **Investigating the implementation impact**. The new design introduces significant change. The effect on the GEF context had to be discovered. The prototype provided valuable insight on which part of the context requires further change and which can be used as is. The prototype also proved that re-use of certain existing GEF parts is possible. This enables a more gradual migration to the new design.
- **Trying out risky parts**. GEF has a number of features that compose its main functionality and form the functional requirements. Those that were most important and at the same time uncertain in the applicability point of view were prioritized for implementation on the prototype. This strategy helped in identifying blocking issues early and putting mitigation plans into effect.
- **Reducing the investment risk**. By implementing the risky and important parts of the new design on the prototype, the underlying problems and effort were revealed. Observations and conclusions about implementation alternatives were noted and this will make it easier for the management to estimate the time effort needed and plan the implementation of the new version of the component.
- **Presenting a tangible result**. Showcasing the new design is equally important with the design itself. The prototype's contribution was very important in this aspect. It allowed the involved stakeholders to have a hands on view of what the new design is capable of achieving.

12.2 What is ASML enabled to do now

This project was the first feasibility attempt that opened the road towards the introduction of a new design for the GEF component. It represents the initial approach to creating a high level solution that addresses the most critical issues of the current GEF. Since this project proved that the proposed solution is feasible, ASML concerning GEF is able to

- Plan the implementation of the new GEF based on the proposed high level design, with higher accuracy and with knowledge of the impact.
- Migrate gradually to the new GEF version. This can be achieved by re-using parts of the current GEF and have a smoother transition until the implementation of the complete Python design.
- Create new GEF clients based on the new design taking advantage of Viper and Python use, following the ASML-specified way of working for CPD tests.

The timeline in Figure 19 depicts what the ASML roadmap is for this component. The change has to go through specific steps. The initial part was the feasibility study that was performed in this project from January until September 2013. In this period the ideal situation had to be discovered, described and also proved. After the end of this phase the green light must be given by the management to incorporate this in a project and implement a functional version of the new GEF also known as realizable situation. Gradually this version can evolve and reach the ideal situation as described by this project.

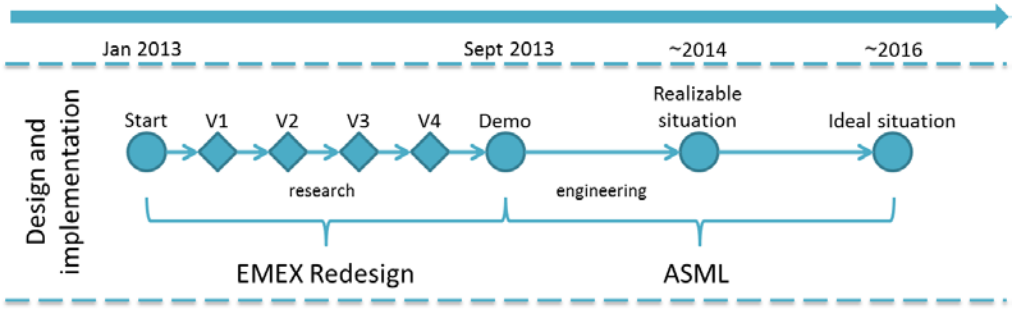

Figure 19 - GEF roadmap

12.3 Future Work

Designing an entire ASML component like GEF requires significant time effort. The feasibility study of this project focused on a design that prioritizes on solving the issues of the current

component and proving its applicability. The ideal situation requires a number of additional decisions to be taken. Some of these were either not applicable in the context of the project or were agreed to be considered at a later stage. Some of these decisions are

- Possible creation of a separate component for the new GEF. Currently the new GEF is hosted in the same component as the existing one. This was done due to re-use of existing implementation and functional similarity. In the future the new GEF can be in its own separate component when the ties with the existing one are not needed anymore.
- Full Python implementation. The new design features a Python based implementation. Use of C is still required though due to incompatibilities of some used components with Python. After these components have been updated to support Python calls, the new GEF can migrate to a complete Python implementation. A down side of the Python migration is that it can introduce double maintenance. So re-use of existing parts of GEF is advised to avoid that.
- Detailed implementation decisions. The limited time of the project did not allow the establishment of all the implementation details. The prototype focused on delivering the important functionality for the proof of concept and not on the quality of the code. Thus, there is room for improvement in that aspect.
- Common GEF exposed symbols. The new design dictates that the exposed symbols of GEF that are used by all clients are forming the common interface part. The symbols that are needed for specific use by a smaller number of clients will be in separate interfaces. The final list of what will be in the common interface and what not is still to be defined.■

13. Project Management

Abstract – In this chapter an overview of the project management techniques used in this project is presented. The initial planning and how it evolved it also discussed. Finally, a list of potential risks for the project is shown along with the mitigation strategies that could be applied.

13.1 Way of Working

The project management approach used in this project was mainly based on Rationale Unified Process [2]. Following RUP, the project period was split into four phases with specific deliverables:

- **Inception**. Problem analysis, requirement definition, project planning.
- **Elaboration**. Solution generation, architecture and high level design evaluated against requirements.
- **Construction**. Detailed design and implementation of the prototype.
- **Finalization**. Documentation and preparation for knowledge transfer.

RUP was used as the backbone of the management process because it was found to fit best with the ASML way of working. The stakeholders involved were more positive in interacting with the process if these phases were clearly defined. This included reviewing documents and attending to meetings based on a linear plan. For the purpose of planning, an Excel file containing sheets for each phase and phase items was used.

Regarding the construction phase, an iteration-based process was used. Since this phase included implementation and try-outs of features on the prototype, a Scrum-like [3] approach was used. The backlog was filled with features which were prioritized and distributed in one month iterations. At the end of each iteration the prototype was presented to the stakeholders. This was also synchronized with the progress steering group meetings which were held once every month and included the company and university supervisors. These meetings were the main reason for choosing a one month iteration length. During these iterations, documentation was also updated according to new findings. The prototype provided significant insight concerning the parts that were not very clear in the Inception and Elaboration phase. A tool for keeping the tasks per iteration and moving them to different columns was used and provided a nice graphical representation of the progress.

13.2 Work-Breakdown Structure (WBS)

The initial WBS for the project was done relatively early in the project. It was an estimate of the time each phase would require and what it would include. Since the knowledge of the domain and the project itself was limited at the time, changes were expected to happen to this plan in later stages. The time estimation for the initial plan was based on the theory supporting RUP. This plan can be seen in Table 7.

Table 7 - Initial project plan

Phase	Purpose	Period
Introduction @ ASML		7 Jan – 18 Jan (2 weeks)
Inception		21 Jan – 8 Feb (3 weeks)
	Problem Analysis	
	Requirements Definition	
	Requirements Review	
Elaboration		11 Feb – 3 May (13 weeks)
	Architecture and High Level Design	
	Alternatives Evaluation	
	Proof of Concepts	
	Design Review	

Construction		6 May – 2 Aug (13 weeks)
	Prototype V1	
	Prototype V2	
	Prototype V3	
	Prototype V4 / Demo	
Finalization		5 Aug – 13 Sep (6 weeks)
	TU/e Report	
	ASML Design document	
Vacation		3 weeks

13.3 Milestone Trend Analysis

After the first meetings concerning the project, it was realized that the expectations of ASML were not very clear. The initial idea was to have a generic design that would fit not only GEF, but more frameworks that follow the same concepts. A significant amount of time was spent investigating different frameworks and components. Effort was put on identifying the causes of the problems that they shared. This already created some delay on the Inception phase.

It soon became clear that the scope of the project was not well defined. After realizing this, the author tried to narrow down the scope and define clearly the expectations of ASML from this project. This led to the decision to focus on GEF and a revised plan was made. Some delay was already introduced from the study of other frameworks. This can be seen in the MTA graph.

One major refinement was made to the plan after the creation of the initial architecture. It became obvious that the time needed for the Elaboration phase was over-estimated. Some decisions had to be implemented in some kind of proof of concept in order to be evaluated. This highlighted the need for more hands on try-outs before decisions could be taken. The Elaboration phase became shorter and an additional iteration was introduced in the Construction phase. This can be seen in the Milestone Trend Analysis graph in Figure 20 with the introduction of the additional iteration and the rescheduling of the first three.

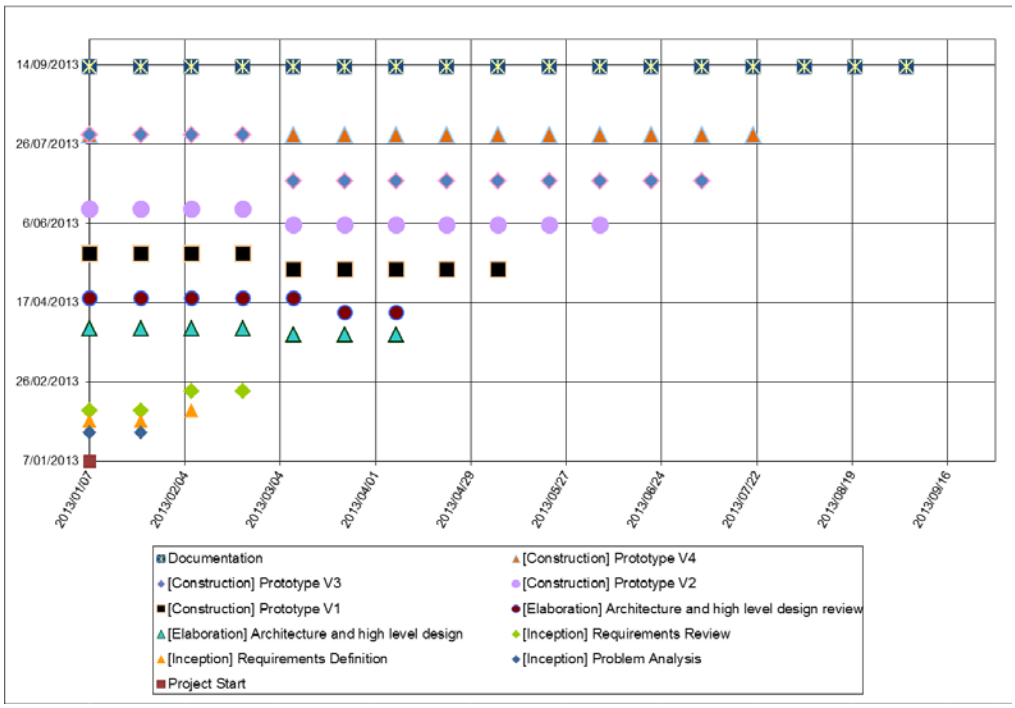

Figure 20 - MTA graph

It has to be noted that although a designated period at the end of the project (Finalization phase) was planned for documents and transfer of knowledge, the document creation started much earlier. The ASML-specific design document had a first version by the end of the Elaboration phase, while the TU/e report was also being written throughout the Construction phase. The Finalization phase made sure that the documents were going to be ready and reviewed by all the required parties.

13.4 Risk Management

During the first period of the project, a number of risks were identified. The most important ones for this project were written down and mitigation strategies were formulated. These can be seen in Table 8.

Table 8 - Potential risks

Risk	Probability	Impact	Mitigation Strategy	Mitigation Effect
Broad project scope	High	High	Negotiate with supervisors to prioritize requirements.	Reduction of the scope.
Lack of domain knowledge	High	Medium	Read documentation and talk to domain experts. For tough domain decisions, ask supervisors.	More information gathered before a decision is taken.
Lack of information resources	Medium	Medium	Create information circle with experts of each technology needed.	Know where to ask for particular information.
Dramatic change of project	Low	High	Negotiate changes. Organize regular meetings to discuss the magnitude of change.	Reduce the effect on the current progress and the time impact.
Insufficient time budget	High	Medium	Negotiate the scope of the project. Focus on “must” requirements.	Ensure the success criteria are well defined.
Viper failure to fulfill all the requirements	Medium	Medium	Negotiate with the Viper team for a solution/extension.	Create a future solution that fulfills the requirement.

13.5 Conclusions

The creation of a plan posed significant challenges in the beginning of the project. The reasons were the initial ambiguity of the project goals as well as the author’s limited domain knowledge. This made accurate estimations very hard. The initial plan was made in order to have a base for the project process and additional refinement had to be done throughout the duration of the project. To conclude, important lessons were learned about the planning process and about the improvement of time and effort estimations while familiarity increases with the domain. ■

14. Project Retrospective

Abstract – In this chapter a reflection on the project is done, looking back into what proved to be good practices and what could have been improved. Furthermore, the design competencies are revisited and their role on the outcome of the project is discussed.

14.1 Good Practices

The project provided a fertile ground to use the knowledge and skills that were acquired throughout the OOTI programme. In addition, it helped the author acquire additional experience in a state-of-the-art environment and being a part of a company that has a leading role in the lithography domain. Some practical rules of thumb were developed during the nine months in ASML.

14.1.1. Frequent Contact with Company Supervisors

Coming into a new environment, especially a very domain-specific one like ASML, can be overwhelming in the beginning. This can be tackled by the help of the company supervisors. In the case of the author the supervisors were almost constantly accessible and it proved to be very beneficial for the progress of the project and the avoidance of blocking situations.

14.1.2. Progress Steering Group Meetings

The meetings with the company supervisors, the TU/e supervisor and the trainee took place once a month. These meetings provided valuable feedback regarding the progress of the project. Furthermore, they were the opportunity to have all important stakeholders at the same table and discuss issues that concern both parties. There are cases that the university and the company had different views. These meetings were the perfect place to clear this kind of issues.

14.1.3. Iterative Implementation of the Prototype

The chosen agile approach for the implementation of the prototype proved to be very useful. The iterations were synchronized with the PSGMs and this allowed the author to showcase what he had been working on each month. It also provided faster results since smaller parts of the functionality were added in each iteration and there were more frequent deliverables.

14.1.4. Technology Learning

The amount of information regarding the ASML domain and software is immense for people new to the company. In the author's case, early experimentation with small learning exercises proved to be very helpful. It allowed familiarization with the domain and the way of working with the ASML system software.

14.2 Improvement Points

When looking back into the project and reflecting upon the experience, there are some aspects that could have been done in a way that would benefit the process more. This retrospective is important and reveals the lessons learned and where more attention can be applied in the future.

14.2.1. Project Planning

As mentioned in the Project Management chapter, the creation of a plan presented difficulties for the author. The main factors for this were the large duration of the project (nine months) and the unfamiliarity with the ASML domain. This made estimations and work breakdown structure less efficient. Since the duration could not be changed, the domain unfamiliarity issue should be attended to more. The investigation for the solution started from a very generic point and then focused on more applicable solutions. If this convergence to more specific

investigations had been done earlier in the project, then the planning could have been more efficient with more insight.

14.2.2. Project Scope and Expectation Management

The expected results of the project were very unclear in the beginning. The scope of the project was broad and covered more than the GEF. This caused significant delay while investigating all the concerned parts. After discussions the scope was narrowed down and that allowed me to focus on GEF. This decision could have been taken earlier so that less time would have been used in investigating other parts. Additionally, the large number of stakeholders and their different points of view created a frequent shift of focus. This could have been handled slightly more efficiently by prioritizing earlier the expectations of each stakeholder.

14.3 Design Opportunities Revisited

In Section 6.5 the design competencies that were deemed relevant to this project at the beginning of the project were described. At the end of the project, these competencies are revisited to determine their fulfillment.

- Genericity: The designed proved that the use of Viper is possible for the purpose of supporting CPD tests in the calibration sequence. The main focus was the fulfillment of the non-functional requirements that can create an efficient base for the needed functionality. This Viper base can be used for other frameworks of the calibration sequence.
- Realization: The feasibility of the solution was proven by the prototype. Although the prototype is not yet a fully functional component, it serves the purpose of providing enough proof that this solution can be beneficial for GEF. Thus this competency remained important and highly influential during this project.
- Complexity: Complexity of the existing GEF was one of the issues that needed to be tackled. This made it an important competency to keep in mind and use for the new design. Indeed, the use of Python assisted significantly towards the direction of reducing complexity and separating responsibilities within the implementation of the new design.■

Glossary

ASML	Advanced Semiconductor Materials Lithography is the world's leading provider of lithography systems for the semiconductor industry, manufacturing complex machines that are critical to the production of integrated circuits or chips.
ASML DSL	An interface definition language describing an interface in language agnostic way. It allows interoperability between different programming languages.
CPD test	Calibration, Performance and Diagnostic test. These tests are used to calibrate, diagnose or measure the performance of a machine during build-up.
IDE	Integrated development environment.
Interface Segregation Principle	This principle states that modules that encapsulate high level policy should not depend upon modules that implement details.
Lithoscanner	Device used in the manufacture of integrated circuits.
Lot Production	The exposure of wafers in large numbers and in long period of time.
MTA	Milestone Trend Analysis
MVC pattern	A software architecture pattern which separates the representation of information from the user's interaction with it.
OOTI	Onwerpersopleiding Technische Informatica
PAS	Philips Automatic Stepper
RUP	Rational Unified Process
Scrum	An iterative and incremental agile software development framework for managing software projects and product or application development.
TU/e	Technische Universiteit Eindhoven
Twinscan	An ASML machine platform that with its unique dual-stage design allows for non-stop processing: measuring one wafer while imaging another.
UNIX	Computer operating system
Viper	Viper is the calibration and performance framework based on Python used by CPD tests in ASML.

Bibliography

1. “ASML website” [Online]
   http://www.asml.com

2. “RUP: Best Practices for Software Development Teams” [Online]
   http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

3. “Scrum software development”
   http://en.wikipedia.org/wiki/Scrum_(software_development)

4. “Multilayered architecture” Wikipedia [Online]
   http://en.wikipedia.org/wiki/Multilayered_architecture

5. “Factory design pattern” [Online]
   http://www.oodesign.com/factory-pattern.html

6. “Life of a function call” [Internal ASML page]
   http://techwiki.asml.com/index.php/Life_of_a_Function_Call

7. “GID User Manual Python and Viper”
   [Internal ASML document ID: 121579]

8. “EMEX vs. EMEXVI Change Impact Comparison”
   [Internal ASML document ID: D000214650]

9. “EMEX Redesign SIA”
   [Internal ASML document ID: D000174630]

10. “EMEXVI Prototype Design”
    [Internal ASML document ID: D000214683]

11. “EMEXVI User Guide”
    http://techwiki.asml.com/index.php/EMEX#EMEX_redesign

12. “Model View Controller design pattern” [Online]
    http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

13. “Design Principles” OODesign [Online]
    http://www.oodesign.com/design-principles.html

About the Author

Spyridon Strouzas received his Diploma in Computer Science and Engineering from the polytechnic department of University of Patras, Greece in 2010. During his studies he focused on software development and architecture. The title of his master thesis was “Location Aware Application for the Android OS” and it involved the design and development of an application which utilized GIS capabilities and social networking elements to allow users to communicate and organize their lives. In 2011 he joined the two-year PDEng Software Technology program of the Eindhoven University of Technology.

3TU.School for Technological Design, Stan Ackermans Institute offers two-year postgraduate technological designer programmes. This institute is a joint initiative of the three technological universities of the Netherlands: Delft University of Technology, Eindhoven University of Technology and University of Twente. For more information please visit: www.3tu.nl/sai.