TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Metrology data modeling and data handling

Citation for published version (APA):
Lukic, M., & Technische Universiteit Eindhoven (TUE). Stan Ackermans Instituut. Software Technology (ST) (2013). *Metrology data modeling and data handling: capturing a domain model of ASML metrology in a software framework*. [EngD Thesis]. Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/2013

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Feb. 2024

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

Metrology data modeling and data handling

Name: Matija Lukic
September 2013

TU Delft
Delft University of Technology

TU/e Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

Metrology data modeling and data handling

Capturing a domain model of ASML metrology in a software framework

M. Lukic

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML Netherlands B.V.

Eindhoven University of Technology

Steering Group

Sofia Szpigiel
Arjan van der Sijs
Ad Aerts

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
978-90-444-1211-6

Abstract
During recent years, ASML metrology functionality has grown intensely. However, the software design principles (such as single responsibility, interface segregation, or open/closed principle) have sometimes been left unattended in favor of time to market. This hampers the implementation of functional design. This report describes the design and implementation of the metrology domain model. The domain model expresses core metrology entities, their attributes, behavior, and relationships. The Onion architecture model and domain-driven design (DDD) characterize the approach towards building the domain model. Since this approach relies heavily on the dependency injection principle, the model becomes a technology-independent core of the software implementation. The results of the project show how the metrology software can look in the future. With the isolated and explicit domain model, software maintainability increases. Moreover, the domain model establishes a ubiquitous language for different engineers, hence bringing the functional design closer to the software design.

Keywords
domain driven design, DDD, embedded software, onion architecture, metrology, ASML, SAI, Software Technology, OOTI

Preferred reference
M. Lukic, Metrology data modeling and data handling: Capturing a domain model of ASML metrology in a software framework. Eindhoven University of Technology, SAI Technical Report, September, 2013. (ISBN: 978-90-444-1211-6)

Partnership
This project was supported by Eindhoven University of Technology and ASML Netherlands B.V.

Disclaimer Endorsement
Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the Eindhoven University of Technology or ASML Netherlands B.V.. The views and opinions of authors expressed herein do not necessarily state or reflect those of the Eindhoven University of Technology or ASML Netherlands B.V., and shall not be used for advertising or product endorsement purposes.

Disclaimer Liability
While every effort will be made to ensure that the information contained within this report is accurate and up to date, Eindhoven University of Technology makes no warranty, representation or undertaking whether expressed or implied, nor does it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks
Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Copyright
Copyright © 2013. Eindhoven University of Technology. All rights reserved.
No part of the material protected by this copyright notice may be reproduced, modified, or redistributed in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the Eindhoven University of Technology and ASML Netherlands B.V.

Foreword

Within ASML, Metrology has the complex task of reaching the nanometer accuracy of ASML's products. Highly-skilled mathematicians and physicists come up with complex mathematical models that describe the physical effects that occur inside the machines. These models are used to obtain the most accurate positions required to expose microchips on wafers. They are implemented in software, together with scheduling and control software. Over the last years the Metrology software grew drastically, as the department size was growing, in order to implement the required functionality. This fast growth produced overcomplicated software, hindering the department's efficiency.

In order to keep developing software for the future, Metrology management decided to start putting more emphasis on the software architecture as a way to improve the quality and efficiency. In this way, a new Metrology Reference Architecture is being designed and proposed by some architects. One of the aspects present in this architecture is the data involved in metrology software and the different mechanisms to store and handle it. Another aspect is the need of bridging the gap between the functional specification and the software design and implementation that is currently in the software. Matija's project was thought to tackle down these problems.

Matija had the difficult task of translating the metrology domain into software, using a completely different approach from the one being currently used. It was not easy, because he only had a few months when usually new employees take around six months to get familiar with ASML's technical concepts. During the nine months of the project, Matija did not only understand the domain quickly, but he also designed and implemented a proof of concept that can show how the new way of working could be. This proof of concept is detailed in the present report.

His speed to deliver added value, and quality of work allowed him to produce valuable results that can help to support the new architecture. We are glad that he decided to stay in ASML after the graduation, and we will be happy to work with him in making the metrology software for the future.

Sofia Szpigiel PDEng.
August 13, 2013

Preface

This report presents the "Metrology data modeling and data handling" project that was carried out by Matija Lukic at ASML Netherlands B.V.[1], Veldhoven, The Netherlands. The project was conducted as a full-time, nine-month graduation assignment in the context of a two-year technological designer program in Software Technology popularly known as the "OOTI" program[2]. This post-master program is offered by the Eindhoven University of Technology under the auspices of the Stan Ackermans Institute.

This project was established as a part of the pilot project for establishing metrology software reference architecture. It focuses on one particular aspect of the architecture, modeling and handling the data. The project fits in ASML's increasing efforts towards applying a model-driven engineering approach in their software development process.

The project was conducted by the author (as trainee) under the guidance of an ASML software engineer who is an OOTI graduate (as a daily company supervisor), ASML group leader (as a company supervisor), and the OOTI program director (as the TU Eindhoven supervisor). Together, this quartet constituted a project steering group headed by the trainee who was fully responsible for the day-to-day management of the project as well as realizing the work described in this report.

The report is intended for anyone who is interested in managing domain complexity in software, particularly in how to develop software for complex domain needs by connecting the implementation to an evolving model. In fact, the design practices in this project are applicable to other domains, especially the ones involving multiple disciplines. Therefore, this report can be interesting also to people who do not know much about metrology. Of course, the capacity for reasoning at a sufficient level of abstraction is assumed.

It is important to mention that this version of the report does not contain any confidential details. The full version of the report is at the disposal of ASML employees and other associates who have signed a non-disclosure agreement with ASML.

The text has been structured in such a way that the reader is smoothly led from problem to solution. The executive summary on page v gives a concise overview of the domain, problem, solution, and results. Furthermore, to facilitate reading, the domain model terms are displayed using this font. Similarly, software-related terms, e.g., methods, parameters, and file names, are denoted using a `different font`. Moreover, a glossary is provided on the page 22 of the document as a reference for terms that may be new to the reader or used in a different context. The first occurrence of a glossary term is indicated by underlining it as illustrated here.

Matija Lukic
September 2013

[1] ASML Netherlands B.V. is hereafter also referred to as ASML or the company.
[2] OOTI stands for "**O**ntwerpers**o**pleiding **T**echnische **I**nformatica." The Software Technology PDEng program is quite popular under this acronym.

Acknowledgements

The “Metrology data modeling and data handling” project could not have been successful without the collaboration of several persons who assisted me in various capacities.

I sincerely appreciate the members of my steering group. I thank Sofia Szpigiel for constantly supporting me with ideas, contacts, tips for handling company politics, and everything that a good supervisor should do. You were always helpful in recognizing risky issues and prioritizing requirements. Moreover, I value our design discussions, many of which find their place in this report. I thank Arjan van der Sijs for always guiding me to deliver the biggest possible business value to ASML. You were stimulating me to think about software redesign not only as a technical, but also as an organizational challenge. Finally, I thank Ad Aerts for providing a non-ASML point of view of the project. Your comments inspired me to think how I could clearly and concisely communicate my ideas to the people unfamiliar with my project.

During my project, I worked with several engineers at ASML. I want to specially thank the following individuals for contributing to my project in ways that these acknowledgements do not permit me to enumerate: Patrick Peeters, Martijn van der Horst, James Downes, Tim Goossen, and John Brusche. I appreciate all your efforts in explaining intricate metrology concepts to me. In addition, I appreciate all my team members who made me feel welcome and answered any questions I had: Jelena Marincic, Ammar Osaiweran, Caizhang Lin, Arjen Klomp, and Sven Weber. Additionally, I would like to thank my fellow trainees who were also at ASML in the same period: Hristina Pavlovska, Panagiotis Leloudas, Umut Uyumaz, Santiago Hernandez, and Spyridon Strouzas. Our regular lunch and coffee meetings helped me relax, take a look at the bright side, and enjoy my internship.

The OOTI program provides tremendous support to its trainees during the program. I express my heartfelt gratitude to my professional development coaches, Cynthia Schreuder and Sandra van Dongen for personal guidance and feedback on the way I presented my project. I thank my technical writing coach, Judith Strother, for her conscientious reviews of this report. Furthermore, I appreciate the advices of other OOTI coaches (especially coming from Angelo Hulshout, Onno van Roosmalen, and Peter Zomer), which were nothing but useful for this project. Finally, I give a special thanks to OOTI Management Assistant, Maggy de Wert, for her prompt responses on my enquiries and her immediacy in communication. All in all, I only hope that I will honor my future work endeavors as much as the OOTI program.

To all those persons who my sometimes transient memory and/or circumstances have foreclosed from these acknowledgements, I would like to say a very big “Thank you,” nonetheless.

Last, but certainly not least, I appreciate my family and close friends for morally supporting me and sometimes justifiably distracting me from my work. Most importantly, I give my deepest thanks to my parents for unquestionably believing in me and enabling me to pursue my dreams outside of my homeland.

Matija Lukic
September 2013

Executive Summary

ASML is the world's leading provider of photolithography systems for the semiconductor industry. These complex machines involve extreme movement of the hardware components with nanometer accuracy (e.g., moving a 15 kg wafer stage at Formula 1 acceleration rates). On such scales, mechanical inaccuracies and geometric distortions are inevitable. Metrology department develops and maintains the software that measures and corrects for these imperfections in the machine.

Over the last few years, metrology has experienced tremendous functional changes and tight deadlines with very high nanometer stakes. As a consequence, the software design principles (such as single responsibility, interface segregation, or open/closed principle) were sometimes neglected in order to have solutions quickly. From the data perspective, two important issues can be highlighted:

1. Using global variables and translating data from one type to another.
2. Assuming a certain state of global variables.

Due to the aforementioned issues, it is difficult to derive the functional design decisions by inspecting the software (i.e., to identify the business logic). This hampers the software implementation and maintenance processes.

This report describes a project to design and implement (as a proof of concept) a metrology domain model. This domain model expresses metrology domain entities, their attributes, relationships, and behavior, as well as domain constraints and invariants. The overall approach towards building the domain model is characterized by the onion architectural model and the domain-driven design. The onion architecture emphasizes the use of interfaces for behavior contracts (i.e., use of dependency injection) and it forces the externalization of technology-dependent functionality. Domain-driven design defines a set of guidelines for modeling domain concepts and evolving the domain model. In general, the domain knowledge was extracted by talking to the domain experts, analyzing design documents and inspecting the current software implementation.

The metrology domain model captures the metrologists' way of thinking, thus bringing the functional design closer to the software implementation. Using the domain model, the ASML metrologists are able to specify the building blocks of metrology functions (such as models, measurements, or sequences) using configuration files. These configuration files are later interpreted by software, which is more efficient than translating functional designs from design documents to software.

Furthermore, the metrology domain model exemplifies how the metrology software can look in the future. Instead of having a scenario-based approach, the software design can rely on state-machines, which map better to how machines react to the events from the environment. In addition, by using the domain model, the size of the code-base can be reduced since the model defines generic, reusable software components. Nevertheless, it is worth mentioning that this proof of concept did not consider throughput-related optimizations of the current metrology software implementation.

The model was verified by recreating real sequences and comparing the sequence execution traces. The traces show that the current metrology functionality can still be achieved with the introduction of a metrology domain model, while improving the maintainability of metrology software. It is recommended to continue working on the domain model, either by refining or extending it. In general, capturing domain models creates a ubiquitous language, embedded in software, for different engineering disciplines.

Table of Contents

List of Figures

1.Introduction

1.1 Context

About ASML

ASML is the world's leading provider of photolithography systems for the semiconductor industry, manufacturing complex machines that are critical to the production of integrated circuits or chips [1]. The vision of ASML is a world where affordable microelectronics improves the quality of life. The way ASML strives to achieve this vision is by designing, developing, integrating, marketing, and servicing advanced systems used by customers – the major global semiconductor manufacturers – to create chips that power a wide array of electronic, communications and information technology products.

ASML produces photolithography machines that deal with optical imaging, a step used in the fabrication of nearly all integrated circuits. In these machines (example shown in Figure 1), predefined patterns are optically imaged onto a silicon wafer that is covered with a film of light-sensitive material (i.e., photoresist). The photoresist is then further processed to create the actual electronic circuits on the silicon. This procedure is repeated dozens of times on a single wafer.

Figure 1. An ASML photolithography machine

The basics of a lithography machine

Looking at it very simplistically, the lithography machine consists of four main components (illustration in Figure 2):

- Light source, which provides light (i.e., laser beam) for optical imaging
- Reticle, which contains the pattern that should be printed onto the silicon wafer
- Lens, which projects the light onto the substrate
- Wafer, a substrate that is coated with light-sensitive photo resist

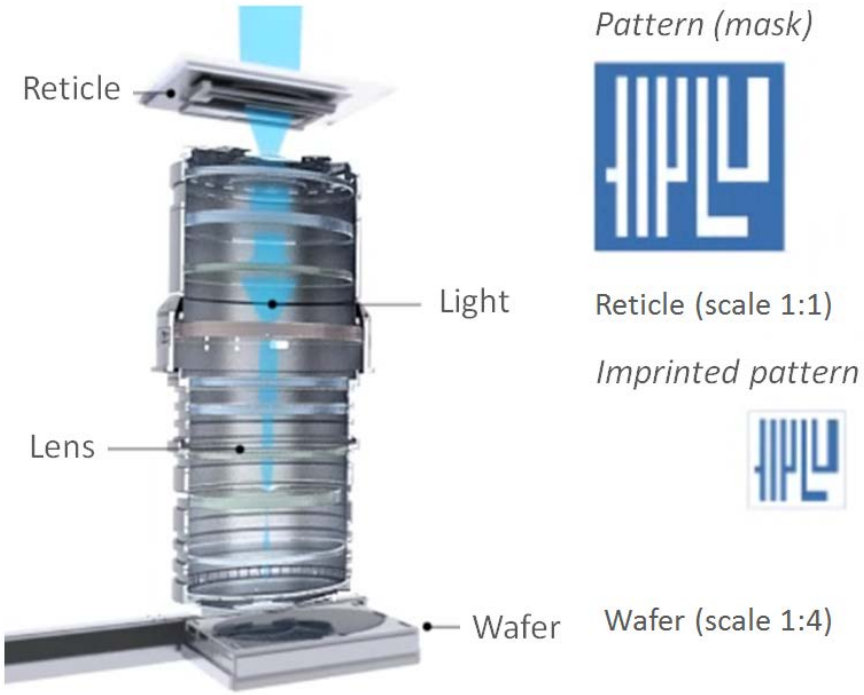


Figure 2. Anatomy of the lithography machine

The primary objective of lithography systems is to project the patterns in the reticle onto the wafer. The source emits the light that hits the reticle. The light that passes through the reticle goes through the lens, which projects the de-magnified reticle pattern onto the wafer. Finally, as the light hits the wafer, features (i.e., the IC patterns) are created on the wafer.

The performance of such systems is characterized primarily in terms of focus, overlay, imaging, and throughput. Focus represents the ability to keep the focal points of the lens as close as possible to the wafer surface in order to get the "sharpest" image possible. Overlay is a measure of how accurately one layer is positioned on top of another layer. Imaging refers to the width of the features that can be created on the wafer. Finally, throughput is measured in terms of the number of processed wafers per hour.

Metrology within ASML

The complex machines are critically dependent on numerous control loop systems to position the hardware components at nanometer accuracy (e.g., moving a 15 kg wafer stage with nanometer accuracy at Formula 1 acceleration rates). Such small-scale accuracy at such high speeds cannot be achieved using mechatronics alone. That is why the Metrology department is needed in order to develop and maintain the software that measures and corrects for the mechanical inaccuracies.

In order to describe the behavior of the machine components, the Metrology department creates mathematical models. Using these models, the imperfections in the machine can be predicted – imperfections such as unwanted drift of elements, distortion due to high-speed movement or intense heating, and aberration of optical elements. The models use the data acquired by sensors to compare the actual state of the machine with the expected (modeled) state and, based on their difference (delta), adjust the machine parameters. In this way, the software constantly corrects for all the physical phenomena that happen at nanometer-level, thus optimizing the overall machine performance.

Optimizing focus and overlay is the objective of metrology within ASML. Metrology does this by adjusting the appropriate parameters of the machine, such as reticle or wafer position, lens parameters, or illumination wavelength. To achieve this goal, metrology

- Sets up the system in such a way that it is brought to a point where overlay and focus errors are minimized.
- Maintains the calibration point during the usage of the machine by defining dedicated measurement schemes and prediction models to cope with various drifts in the system.

The focus and overlay errors are minimized when the aerial image (i.e., projection of the reticle onto the wafer, see Figure 3) coincides with the surface of the wafer.

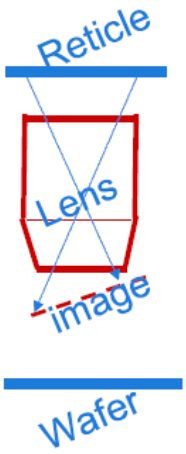


Figure 3. Aerial image

In order to increase throughput, a TWINSCAN is divided into *a measure* and *an expose* side. The division allows parallel processing: one wafer is measured on the measure side while another one is exposed on the expose side. Metrology is responsible for both sides of the machine.

1.2 Assignment objective

Even though the theory and the algorithms developed in the metrology department are advanced, their software implementation suffers from the typical problems that can be observed in large, evolving software systems. Examples include different coding styles, as well as documentation and code not being in sync.

Furthermore, the geometric aspects of the machine are modeled in software using loose parameters[3] or C structures. In many cases, these parameters reside in a "global context," which is accessed from various places in the software.

The underlying problem is that the data-dependencies and ordering of the models are not always expressed in the software, thus making it challenging to understand the data-flow through the system. Moreover, the software components have a large number of inter-dependencies that are a result of suboptimal system decomposition.

The task of this project is to create a software framework that will capture the metrology domain concepts, with the aim of bringing the functional design closer to the software design. By providing a robust domain model, it will be possible to simplify the software implementation, making it more maintainable and extensible. In addition, the framework should provide an API via which its clients will be able to create, parameterize, and query domain entities.

2.Problem Analysis

2.1 Metrology software

Within the TWINSCAN software stack, the metrology software adjusts the machine state for exposing the wafers, by accounting for the physical imperfections in the machine. For instance, a wafer surface at nanometer level is never flat and it contains bumps. Metrology can adjust the focal point of the lens to accommodate for these changes in the wafer height and thus the system will be able to properly expose the wafer.

[3] Loose parameters do not clearly express their intent. Primitive data types and pointers are often used for such parameters.

Since it is impossible to measure everything in absolute terms on a nanometer-scale, metrology uses a number of reference points to represent the spatial information in relative terms. Additionally, the spatial information is expressed in different coordinate systems. Metrology software transforms positions from one coordinate system to another.

2.2 Metrology software issues

In the last couple of years, metrology has experienced tremendous functional changes and tight deadlines with very high nanometer-scale stakes. As a consequence, the software design principles (such as single responsibility, interface segregation, or open/closed principle [2]) were sometimes neglected in order to have solutions quickly. Furthermore, the number of employees has doubled in a relatively short period of time. From the data perspective, two important issues can be highlighted:

1. Arguments between the function calls are often passed using global variables. Moreover, since every component defines data in a different way, the data needs to be transformed from one type to another.
2. The functions in the software implementation often assume a certain state of global variables and these assumptions are hard to track. Furthermore, due to the lack of a proper domain model, the software implementation does not correspond to the functional design.

Due to the aforementioned issues, it is difficult to derive the functional design decisions by inspecting the software (i.e., identify the business logic), which hampers the software implementation and maintenance processes.

2.3 Stakeholders

There are four types of stakeholders who have an interest in this project, each one having a unique set of interests. The stakeholder groups are depicted in Figure 4.

ASML management members are the big-decision makers within ASML who develop the long-term plans and roadmaps for the TWINSCAN machines. The project results will be presented to them as a selling point for increasing the efficiency of the software engineers by redesigning the metrology software. If satisfied with this proof-of-concept, the management will support further metrology software refactoring and migration towards more modern technologies and processes.

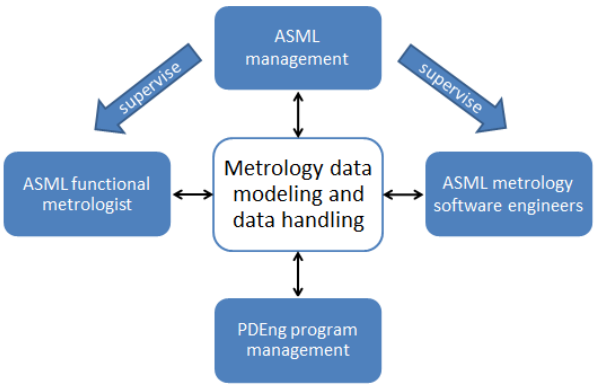


Figure 4. Project stakeholders

ASML functional metrologists are responsible for creating and often implementing mathematical models that describe machine behavior and all the physical phenomena that happen inside the machine. They will speed up their work by using the software framework, instead of having to write all the code themselves. Consequently, the process of metrology model implementation will become less error-prone.

ASML metrology software engineers are developing and maintaining metrology software components, sometimes without having the expert knowledge of metrology models. Their way-of-working would benefit from having software that captures the knowledge of metrologists, so that they could better understand, maintain and enjoy constructing the metrology software.

Management of the PDEng program in Software Technology ensures that the project meets the requirements to grant the PDEng in Software technology degree. Their interest is that the PDEng candidate shows a high-quality design-skillset during the project. In addition, the university also wants to use the project outcomes to promote the program towards students and companies.

2.4 Main challenges and project approach

The main challenge in this project is to provide the functional metrologists with domain concepts that capture their way of working, while at the same time providing a redesign of the software that is understandable for metrology software engineers. They should be able to maintain it and extend it, while guarding the integrity of the concepts functional metrologists use.

Another challenge is to cope with the dispersed knowledge of the metrology domain. There is no single source of information on metrology that could be used for extracting the specifications. Instead, local experts for certain parts of the domain were interviewed in order to extract the domain knowledge. However, as it often is the case in large companies, people with a lot of domain knowledge are very busy.

Lastly, modeling the whole metrology domain within the timeframe of this project would hardly be an achievable feat. Therefore, the scope was set smaller at first and it was later expanded. Initially, the focus was put on the core of the domain and only once the core was modeled, some smaller model refinements were done. It was preferred to have a smaller number of domain concepts that are modeled properly rather than to have a large number of domain concepts that might not reflect the metrology domain in the best way.

Due to the challenges that this project brings and the proof-of-concept nature, an iterative and evolutionary approach was applied. The first month project was dedicated to get familiar with the working environment and define the scope of the project. After that period, the work was done in biweekly iterations (cf. Chapter 7).

At the beginning of an iteration, a task list for the planned features was created. This was done with the project supervisor, with whom a close contact was kept by means of daily communication on progress and important design decisions.

At the end of an iteration, a session with the project supervisor and some of the stakeholders was held in order to gather their feedback and discuss features to be implemented in next iterations. In these sessions, the main focus was on the functionality and not the design.

There were a large number of unknowns: the behavior of metrology software or reasoning behind some design decisions, entangled software control flow, constant mixing of abstraction levels in software, and so on. In such a situation, where it was not known up-front what the final product should look like, the evolutionary approach proved to be useful. By making prototypes, revisiting requirements and applying design-for-change principle, the analysis paralysis was avoided.

2.5 Design opportunities

Kees van Hee and Kees van Overveld had defined criteria for assessing a technological design in a report from March 2010 [3]. In this report they outline nine design aspects (i.e., criteria) that are relevant for a design. For this project, three of those were picked out as the important ones for this project. In addition, two design criteria

were identified as not relevant to the project. These relevant and non-relevant criteria were selected at the beginning of the project and they are analyzed in this section. At the end of the project, these criteria were reflected upon and Section 8.2 explains how the quality of the design was improved for the important criteria.

The design criteria that are of interest for this project are

- Genericity
- Functionality
- Complexity, in particularly the reduction of complexity

Genericity is a vital design criterion for this project, as the project is a step in the development of the metrology software reference architecture. This project focuses on the data repositories that are used by a number of other software components. Therefore, the design has to be generic enough to support different scenarios and different data types, as well as abstract from any technology specific aspects. For that purpose, interfaces that do not reveal implementation details have to be defined. In addition, since data definitions are not reused as much as they could be, this project aims to improve that aspect by providing data sharing mechanisms.

Functionality is another design criterion relevant for this project. As mentioned, the outcome of this project (i.e., software framework that captures the metrology domain model) will be used by other software components. The new reference architecture is a brown-field type of project, which means that it focuses on redesign. As such, the architecture has to support all the functionalities that exist currently in the software, and the data repositories layer is no different. Data entities in the metrology software have to be identified and incorporated in redesigned components.

Complexity is the last of the three selected design criteria. The whole effort in redesigning the metrology software aims to reduce the software complexity, by capturing the domain concepts in software. Currently, the definitions of the metrology data objects are very loose, which makes the process of constructing software needlessly tedious and error-prone.

The design criteria that are not of interest for this project are

- Impact
- Inventiveness

Impact is not taken into consideration for this project because the produced artifacts are prototypes by nature. In other words, the artifacts will not be deployed on a real TWINSCAN machine within this project. There is no chance of incidents happening nor is there a considerable social impact. However, after this project is done, the economic value and the sphere of impact will become more relevant as the metrology software reference architecture is rolled out and changes the way the Metrology department is working and thinking about its domain.

Inventiveness is another design criterion not emphasized during the project. This is because the domain is already known by metrologists and this project focuses on converting their tacit knowledge to explicit knowledge. Thus, no implemented concepts should be surprising for the metrology domain experts. Also, due to the limited time for this project, existing software solutions are used to retain focus on the metrology domain modeling as much as possible.

3.System Requirements

3.1 Requirements management

The technique used for the requirement prioritization is MoSCoW [4]. This technique is commonly used in software development to reach a common understanding with

stakeholders on the importance they place on the delivery of each requirement. The name of the technique is derived from categories used:

- *MUST*: Describes a requirement that must be satisfied in the final solution for the solution to be considered a success.
- *SHOULD*: Represents a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary.
- *COULD*: Describes a requirement that is considered desirable but not necessary. This will be included if time and resources permit.
- *WON'T*: Represents a requirement that stakeholders have agreed will not be implemented in a within the scope of the project, but may be considered for the future.

3.2 High-level requirements

The high level requirements represent the most general requirements for the project. In order to understand the requirements correctly, it should be clear what is meant by two similar terms which might be confused with one another:

- *Domain model* is a conceptual model that describes the various entities, their attributes, behavior, roles, and relationships. In addition, it specifies the constraints, rules and invariants that govern the problem domain. Implementing a domain model in software implies capturing all the specifications of the domain model in a software framework.
- *Model* is an executable mathematical model (i.e., an algorithm) that improves the knowledge of the deviations in the machine based on the actual and predicted state of the machine.

Must have

1. Create a domain model, with the focus on the exposure side of the TWINSCAN.
2. Create a software framework for defining and parameterizing metrology domain concepts.
3. Create and maintain persistent data repositories with defined data contracts for the shared data.

Should have

4. Isolate the technology-dependent parts of the implementation.
5. Provide interfaces to integrate the framework with the existing software components and create local tests with stub interfaces.

On a smaller scale, the *should-have* high-level requirements must be done, meaning that it is unacceptable that they are completely ignored.

Could have

6. Deploy and test the code on ASML's test platform.
7. Generate code for the data entity interfaces using some modeling tool.
8. Extend the domain knowledge with measure-side domain.
9. Integrate the code with behavioral components (i.e., higher level controllers).

Won't have

10. Develop a scheduler that executes the sequences.

4.System Architecture

4.1 Architectural approach

In a business setting, it is important to capture business rules in software. The three-tier architecture is a traditional approach based on SOLID [2] principle of separation

of concerns. However, in the long-run, it falls short when it comes to software evolution and that is where the onion architecture provides a better approach.

Three-tier architecture

For the purpose of developing business applications, one of the most prominent architectural patterns is the layered architecture [5, p. 19]. The layered architecture is traditionally implemented as a three-tier architecture in which the user-interface, business logic, and data storage are developed as independent modules. Each layer depends on the layers beneath it (as can be seen in Figure 5) and then every layer will normally depend on some common infrastructure and utility services (e.g., messaging middleware or persistency solutions).

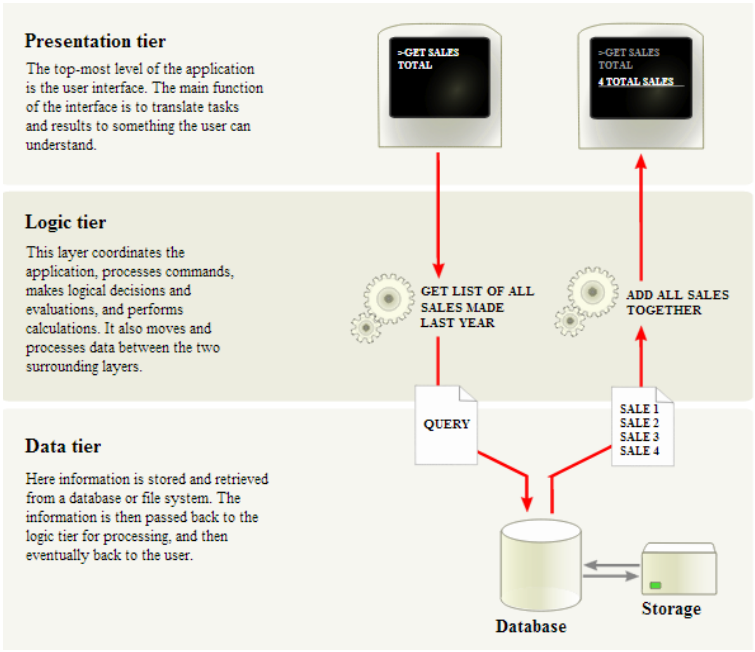


Figure 5. Traditional view of the layered architecture

A common characteristic of systems developed using this approach is tight coupling of the domain model (i.e., domain entities and behavior) with the data storage technologies. In some cases, this does not represent a drawback and it might significantly reduce the implementation effort.

However, in case of long-lasting business applications, the business logic and the data model evolve over time. Furthermore, new data access techniques are developed by the industry every couple of years (recent examples are NoSQL and REST) [6]. Hence, it is likely that the data access requires modification for any long-lived system. Still, if system upgrade is hampered by the coupling, then the business has no choice but to let the system fall behind. In this way, the legacy systems become outdated, and eventually end up being completely rewritten.

In ASML, this is exactly the point where metrology software is currently at. A substantial effort is required for adding new functionalities to the metrology software. The effort could be reduced with a universal domain model which captures the essentials of metrology business.

Onion architecture

The onion architecture takes another view of the system, placing the core of the business in the center. The dependencies can go towards the inner circles of the onion model and the inner circles are unaware of the outer ones. This architecture model relies heavily on the dependency injection principle [6].

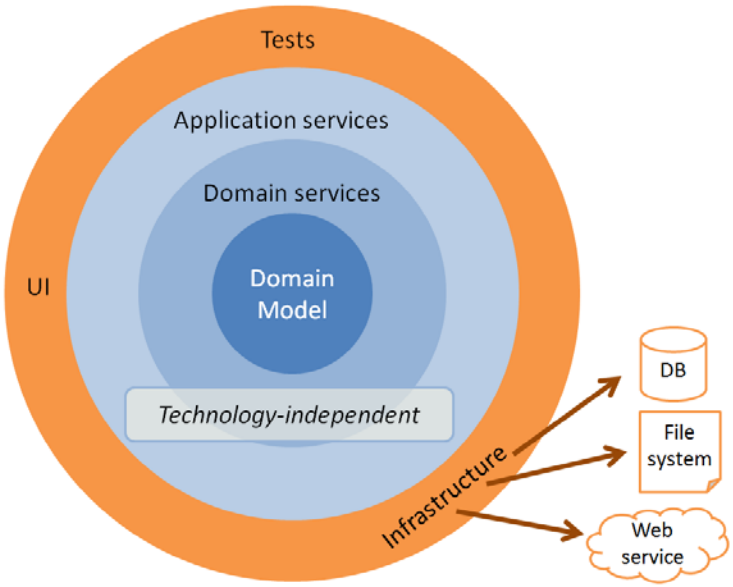


Figure 6. Onion architecture model

As it can be observed in Figure 6, at the heart of the application sits the Domain Model, which captures the important entities and relationships. The domain model does not have any external dependencies; it depends only on itself. Interfaces for accessing the domain objects are also defined in the domain model. However, their implementation is dependent on the data storage and as such is not a part of the domain model, but the infrastructure, which is the outermost ring.

Application/Domain Services help to define the behavior and the intent of the application. These services, when implemented, orchestrate calls to the repositories and return data to the client. The domain services capture the business rules of the domain. The application services, built on top of the domain services, provide all the necessary functionality for handling client's requests.

Comparison of architectural approaches

The biggest difference between the three-tiered architecture and onion architecture is in how infrastructure code is treated. In three-tier architecture, a database scheme often captures a data model. With the onion architecture applied, there are no "database applications" because the core domain model is not encapsulated in a database scheme, but in the software itself. Decoupling the application from the databases lowers the cost of maintenance since the domain model can be changed independently of the infrastructural layer.

In metrology, the onion architectural style can prove to be very useful, as it allows the functional design to become the centerpiece of the software. In that case, the gap between the functional metrologists (i.e., metrology domain experts) and software engineers can be bridged. The business logic is isolated from the data storage and the logical action layer, thus making metrology software easier to understand, maintain and further develop.

4.2 High-level decomposition

Following the Onion architecture, four components were developed during this project. These components are depicted in Figure 7.

Domain contains the domain entities and services. It captures the domain knowledge of metrologists. This layer is completely independent of technologies and it is meant for general usage.

Application contains prototypes of the business logic entities. It exemplifies how the domain model can be used in order to codify the business logic of metrology (e.g., which scan can be performed using which sensor or what should the execution order of the subsequences be)

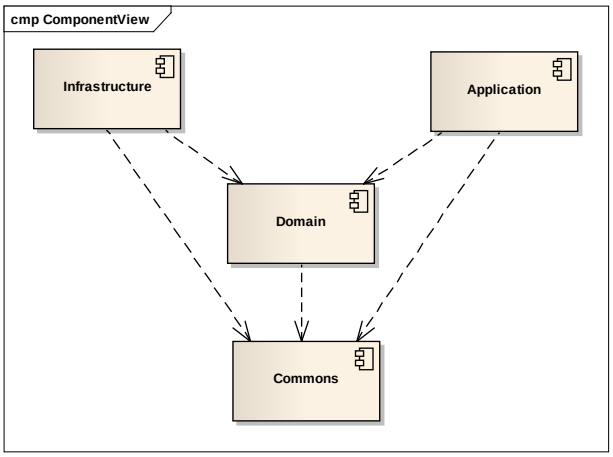


Figure 7. Component diagram – high-level decomposition

Infrastructure contains all technology-dependent functionality. This layer also uses the domain layer. This is the main difference compared to the layered architecture – the dependency is reversed. In other words, all components that depend on the domain model and external components are considered as infrastructure components. An example is a repository that stores domain objects in a database.

Commons represents a set of general interfaces and utility functions. This layer does not contain anything that is specific for the metrology domain model. It can be viewed as a programming language extension, e.g., basic interfaces such as repositories and events are defined there. All other layers use the interfaces defined in this layer. In general, this layer can be reused for building other domain models as well.

Design principle: dependency injection

The domain model does not depend on any technology-dependent services. By relying on the dependency injection principle [2], it allows the client components to specify technology-dependent entities in the domain model. However, these entities have to implement some of the common interfaces. An example of the dependency injection principle is related to repositories, as shown in Figure 8.

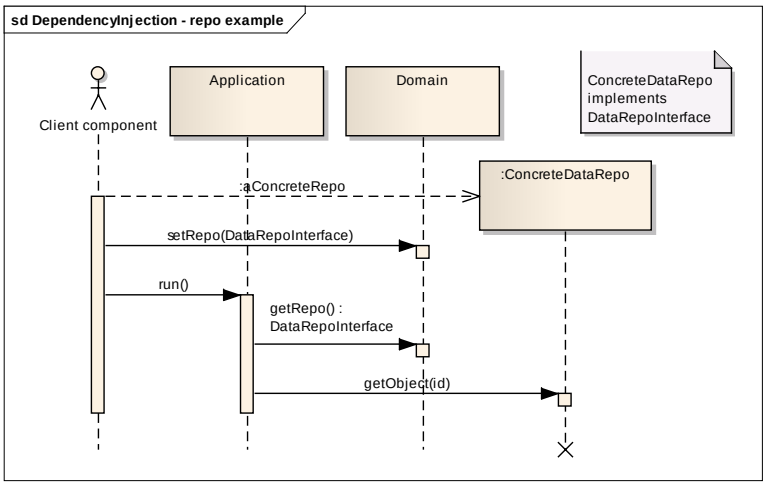


Figure 8. Example of dependency injection – repositories

The *Domain* offers methods to get and set a repository for some of the domain objects. Each repository has to implement a common repository interface. The domain does not know anything about the repository implementation.

Initially, the *Client component* creates a concrete repository (defined in the *Infrastructure*) and sets it in the domain model. Afterwards, *Application*, when using the domain, gets the previously set repository.

The *Application* is also aware of the common repository interface only and it is kept free from any concrete repository implementations. The choice of the implementation is up to the component that initializes the system, which allows better testability and maintainability of the software.

4.3 Domain driven design

After putting the complexity of the domain in the heart of software using the onion architecture, the next logical step is applying domain driven design (DDD), a collection of principles and patterns that help developers craft elegant systems. Properly applied, DDD can lead to software abstractions that accurately express domain knowledge [7]. These abstractions encapsulate complex business logic, closing the gap between business reality and code. A collection of such abstractions is referred to as the **ubiquitous language.**

From the start of this project, it was noticed that there is a lack of the ubiquitous language among the metrologists. Examples of ambiguity include the following

- *Scan* and *measurement* were used as synonyms even though a measurement defines the parameters based on which a scan is performed.
- *Scan* was used for any kind of action that is done by the lower subsystems which involves sensors, whether it is exposing a wafer, measuring or simply busy-waiting for conditions to be appropriate.
- There was no concept of transformation state parameter accuracy or the events that affect it.

Upon discovering these ambiguities, the need for building the metrology domain model was evident and domain-driven design was selected as the approach to do so.

Building blocks

Domain-driven design defines a number of basic building blocks that help identify the properties of domain objects easily [7, pp. 81-163]. These building blocks, used to form a domain model, are the following:

- **Entity** is a domain object that has a unique identity and a defined lifecycle.
- **Value object** is a shared, immutable domain object that does not need an identity.
- **Aggregate** is a collection of objects that are bound together by a root entity, otherwise known as an aggregate root. The aggregate root guarantees the consistency of changes being made within the aggregate by forbidding external objects from holding references to its members.
- **Service** is an operation that does not conceptually belong to any object. It provides behavior to be used by the domain objects or the client application.
- **Repository** is a place that holds the current state of a certain type of an aggregate. It mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.
- **Factory** is a domain concept that contains rules for creating aggregates.
- **Domain event** is a notification about a change in the domain (e.g., updated state of an aggregate). It is a convenient modeling primitive for event-centric systems such as a lithoscanner.

5.Design and Implementation

5.1 Metrology domain model

The metrology domain model captures all the metrology domain entities and their relationships. Figure 9 shows the core of the domain model.

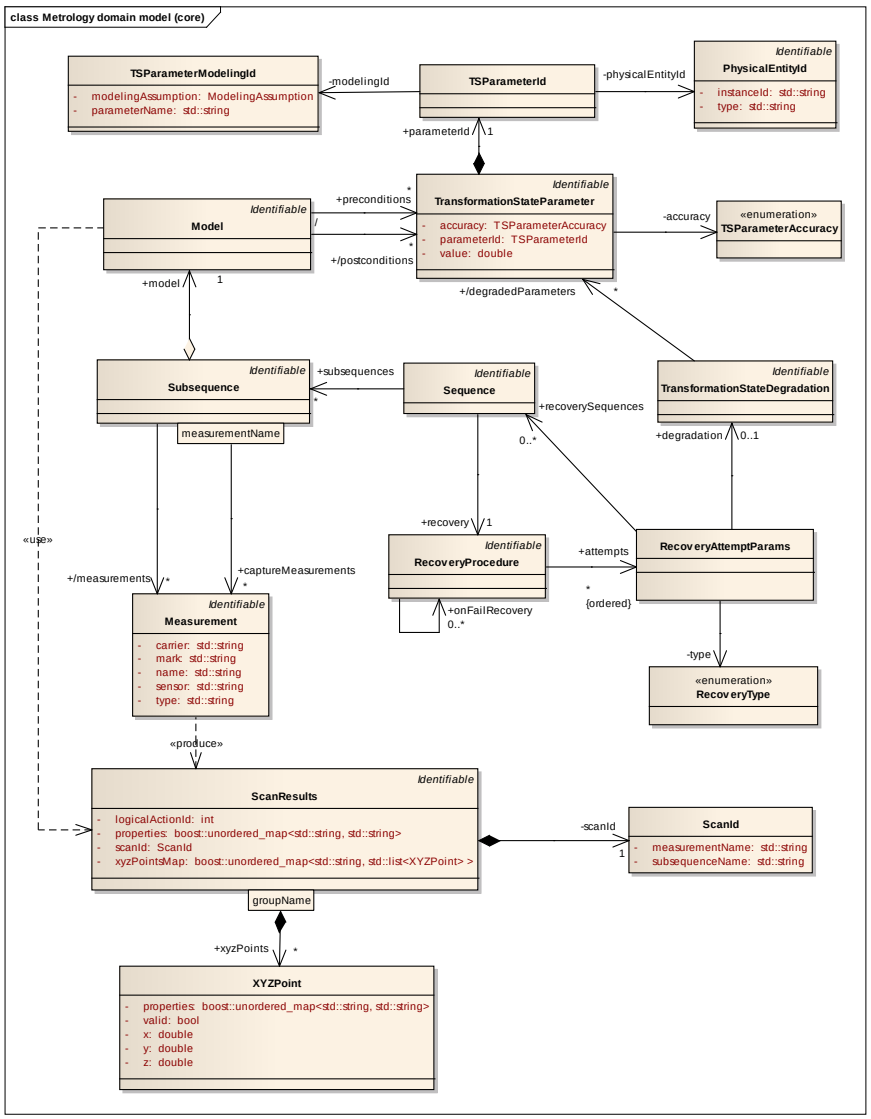


Figure 9. Class diagram – core of the metrology domain model

TransformationStateParameter is an entity that describes a certain geometric aspect of the physical entities in the machine. All parameters in the system together form a transformation state. The corresponding identifier is the TSParameterId value object.

Model is an entity that improves the knowledge of the geometric aspects in the machine based on the state of the machine. A Model is identified by its name.

Measurement is an entity that defines a scan for obtaining information about some PhysicalEntities in the machine. An example of such information is the current position of a PhysicalEntity. A Measurement is identified by its name.

Subsequence is an aggregate root that bundles a Model with a number of Measurements. The results of the Measurements are used as inputs for the Model.

By executing a Subsequence, the transformation state accuracy can be increased by a certain degree. A Subsequence is identified by its name.

TransformationStateDegradation is an entity that represents events that reduce the knowledge of the geometric aspects in the machine. If a TransformationStateDegradation is activated, the transformation state accuracy is decreased to a certain level. A TransformationStateDegradation is identified by its name.

Sequence is an entity that contains a set of Subsequences. By executing a Sequence, the transformation state accuracy is increased step-wise to the highest accuracy. In case of an error, an associated RecoveryProcedure is executed. A Sequence is identified by its name.

ScanResults is an entity that contains a number of XYZPoints and a set of properties. They are a result of performing a certain Measurement. A Measurement can be viewed as a scan definition, and a scan is an instantiated measurement, by a certain subsequence and at a certain moment in time. ScanResults entity is identified by a ScanId.

RecoveryProcedure is an aggregate root that consists of a number of recovery attempts. The recovery attempts can be of different types, some of which include executing recovery-dedicated Sequences and ing TransformationStateDegradations. In case of an error, the next one in a recovery-chain can be executed. A RecoveryProcedure is identified by its name.

PhysicalEntity is an entity that describes hardware elements (parts of the machine) or the materials (used by the machine). A PhysicalEntity is identified by a PhysicalEntitiyId**.**

5.2 Usage scenarios

In the process of system design, the main components are identified by considering the use cases for the system. Examining the usage scenarios enables the designer to identify the responsibilities of system components. The same approach was taken for building the metrology domain model. The model was examined and expanded incrementally to capture all the scenarios depicted in Figure 10.

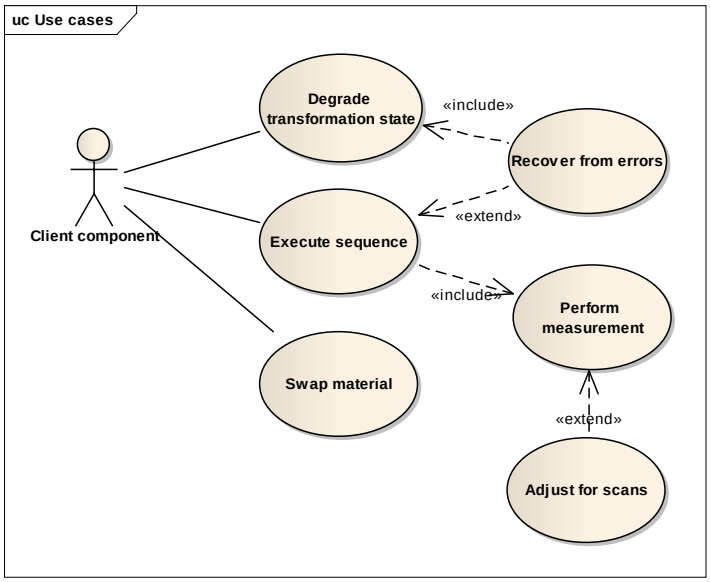


Figure 10. Use cases for the metrology domain model.

Here, a *Client component* is in the role of a user for this domain model, but in general it can be any client of the library that contains the metrology domain model.

Executing a sequence

The main usage scenario is executing a Sequence, in order to bring the transformation state to a certain, higher, accuracy level. Executing a Sequence includes executing all its Subsequences. A Subsequence is executed by performing its Measurements and executing its Model.

Degrading transformation state

Sometimes, in a machine, it can happen that the knowledge of the geometric aspects in the machine becomes invalid. In case of such events, the transformation state needs to be degraded, i.e., the transformation state accuracy is downgraded.

Performing measurements

Measurements are performed by moving the hardware elements and reading the sensor. For the scope of this project, the interface for scan execution essentially consists of two methods:

- Initiating a scan, given a certain sensor, type of scan, and a mark.
- Obtaining results, given an identifier of a previously initiated scan.

Recovering from errors

Performing scanning or modeling actions can be unsuccessful. Whenever a Sequence fails, its RecoveryProcedure is executed. Recovering from an execution failure means that several recovery attempts are made in succession. Each attempt can deploy different techniques to recover from errors.

Swapping material

TransformationStateParameters describe the geometric aspects of the machine. If a new material coming to the machine (e.g., reticle), the current knowledge of geometric aspects becomes invalid. If a material is returning to the machine, the previous knowledge of geometric aspects can be restored.

Adjusting for scans

There are certain conditions that need to be fulfilled before executing a scan in order to avoid getting an invalid sensor reading. An example involves activating a sensor:

- Prior to the scan execution, a sensor used in the scan should be activated.
- Once it is no longer needed, a sensor could be deactivated.

5.3 Package distribution

The classes in the `Domain` layer are grouped in a number of packages, according to their purpose. This layer is in the center of the onion architecture and therefore it is independent of any other layers. All packages and their mutual dependencies are shown in Figure 11.

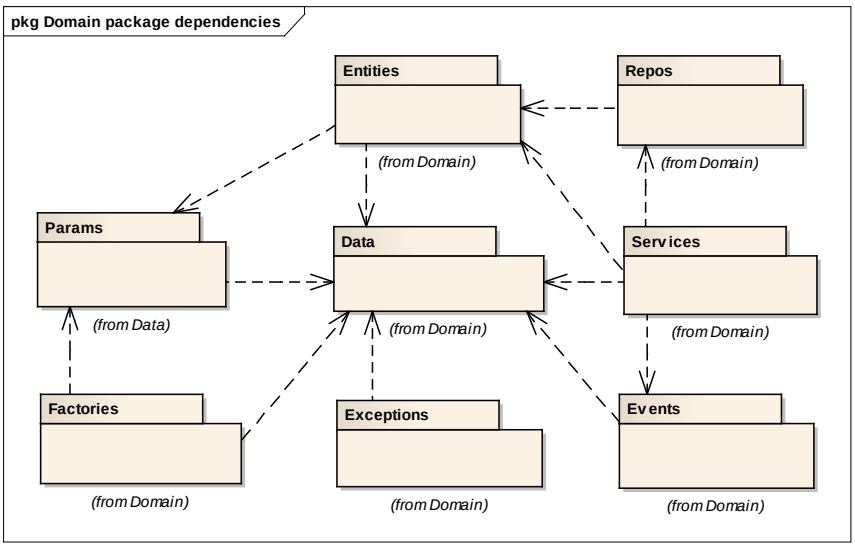


Figure 11. Domain package dependencies

`Entities` package contains first-class citizens of the domain model that capture the main metrology concepts. Entities are primarily data-oriented, meaning that they do not possess so much behavior. They depend on the parameters from `Params` for initialization and value objects from the `Data`.

`Params` package contains parameters for the entity initialization. As the entities are read only, in order to avoid declaring friend-classes, a set of parameter classes was implemented.

`Data` package contains identifiers, enumerations, simple aggregates, and other value objects that domain entities refer to. The content of the Data package is also used by the domain services.

`Repos` package contains repository definitions. Since all repositories work with aggregates, this package depends on the `Entities` and `Data`.

`Events` package contains all the domain events. The domain events, using domain data, spread the information about the changes across the domain.

`Exceptions` package contains definitions of exceptions that are thrown whenever execution fails.

`Factories` package contains a number of utility classes that can be used to create domain entities or data. They capture instantiation rules of the domain.

`Services` package contains behavior-oriented classes that implement domain rules, constraints and other functionality. They use practically all other packages in the Domain layer, because they are responsible for changing the state of the domain and committing those changes to the repositories.

6.Conclusions

6.1 Project results

Domain model as ubiquitous language

The main objective of this project is to bridge the gap between the functional design and the software implementation. By using concepts defined in the metrology domain model, the functional metrologists can still specify machine functionality. However, now they can specify it in terms of building blocks that can be directly plugged into the software. If software is constructed to reflect the metrologists' way of thinking, anyone inspecting the software will be able to derive what the software should do. The domain model becomes one **ubiquitous** language, used for communication between different metrology experts. For functional metrologists, this would mean avoiding implementation details when specifying a functional design. For software engineers, this implies explicit software requirements that can be translated to code more easily.

Next-generation metrology software

Another objective of this project is to show what the metrology software can look like in the future. The prototype of the metrology domain model can give a glimpse into the future of metrology software:

1. The software can be based on state machines and events instead of procedures and sequences of actions. Procedural thinking is more natural for humans, but machines operate by reacting to events from the environment.

2. The size of the code base can be reduced by utilizing object-oriented mechanisms (inheritance, polymorphism, interfaces). Furthermore, proper system decomposition allows the creation of generic components that can be reused, replacing repeated segments of code.

Other results

Aside from exemplifying how metrology software could look in the future and bridging the gap between functional and software design, a couple of other aspects valuable for ASML were done during the project:

1. The usage of databases for metrology software was examined (which type of database, what kind of querying is required).

2. The metrology domain model was fully documented as it gradually grew. In addition, other insights about the metrology software (the ones not strictly related to metrology domain model) were noted down. These insights were extracted from the talks with the domain experts. The documentation can be used for the future metrology newcomers who need to get their head around the fundamental metrology concepts.

3. Alternative data handling techniques that can be used in metrology were explored.

6.2 Future work

This project was aimed at exploring and showing design concepts that can improve metrology software. Because of the limited time-frame for this project, the primary focus was capturing the metrology domain knowledge and building the domain model prototypes. With that in mind, given the maturity level of the developed domain model prototypes, there are three possible directions in which the results of this project can be used:

1. Other domain models that use (or can be used by) the metrology domain model can be developed. These models can have the same level of abstraction as the metrology domain model, but they reside higher (or lower) in the software layering stack.

2. Other components can use the metrology domain model to implement control logic for executing sequences. These components can capture the concrete metrology business logic.

3. The metrology domain model can be refined further. These refinements can result in a domain model that maps the existing software better when implementing new concepts and features. The latter can help the functional metrologists to align the functional designs with their way of thinking.

7.Project Management

7.1 Project planning and scheduling

The project can be divided in roughly four phases (in braces is a mapping to activities depicted in Figure 12 and Figure 13)

1. Learning (1-4)
2. Prototyping (5-6)
3. Model consolidating (7-8)
4. Documenting and closing (9-15)

At the start of the project, in the learning phase, the emphasis was on understanding the basic metrology concepts and getting educated on the ways of working in ASML. In addition, at this stage, the problem was defined more precisely and a set of requirements was devised.

After getting to know the problem at hand, in the prototyping phase, the domain model was built incrementally. By taking one usage scenario at the time, new concepts were specified and the domain model was extended. The prototyping was tackled in biweekly iterations.

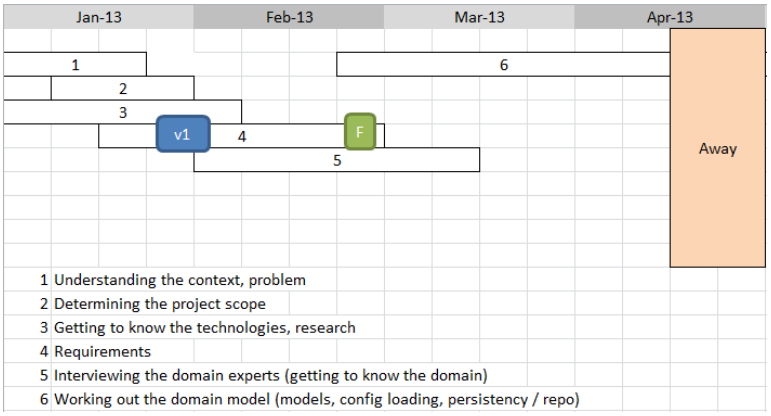


Figure 12. Overall project plan (1/2, Jan-Apr)

Upon building the domain model to a good measure, in the consolidation phase, it was time to devote attention to the more technical aspects of the solution. This stage included amongst other refactoring, changing the coding style to adhere to ASML standards and more thorough testing and adding some auxiliary features to the domain model.

Finally, with the domain model in place, in the documenting and closing phase, the emphasis was on finalizing the reports and other deliverables. In addition, the metrology domain model was disseminated throughout ASML.

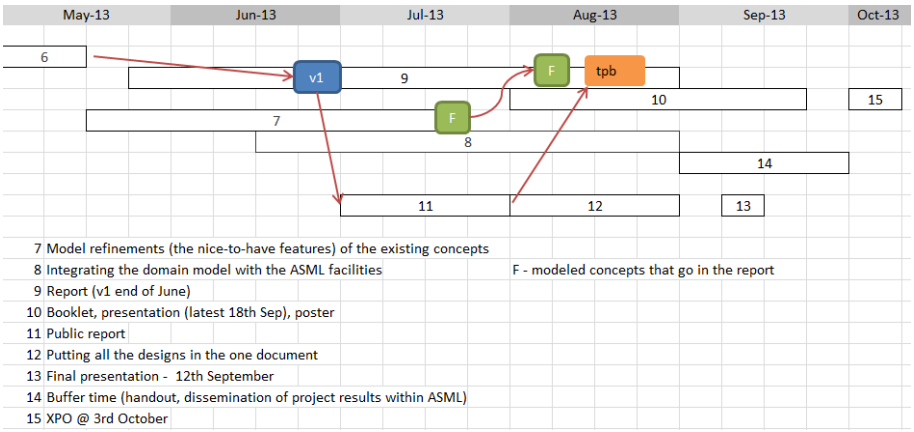


Figure 13. Overall project plan (2/2, May-Oct)

The plan was presented to the supervisors after the first month. Afterwards, it was periodically revised and any changes made were presented again.

Every month a more detailed plan was presented to the supervisors during the project steering group meetings. Additionally, an overview of last month's activities was given. Important decisions were taken during these meetings, especially regarding the deliverables, and they were always followed up by sending the minutes of the meeting.

At the end of the second month, after the initial learning phase where the problem was defined, the requirements were discussed with the supervisors. In the course of the following few months, the domain model was built incrementally in seven iterations. After that, together with the technical team lead and the direct supervisor, the integration options were discussed. Once the choice was made (together with the supervisors), the project entered the consolidation phase.

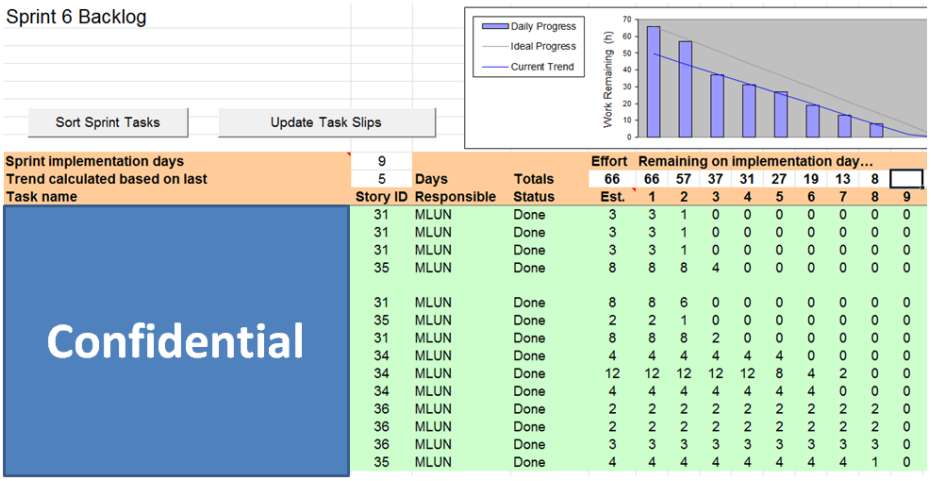
During the project, the documentation of the domain model was progressing steadily. In the first half of the project, there was a constant pace of writing one report chapter per month. With the domain model consolidated, the focus was put on completing the documentation and the reports were completed a month before the end of the project.

Finally, the time left was used to prepare for the final presentation and to tie up any loose ends.

7.2 Work-breakdown

Throughout the whole project, the work was done in iterations of two weeks. At the beginning of each iteration, the planned features were broken down to smaller tasks (example in Figure 14), together with the supervisor. The time estimation was done for individual tasks and the remaining hours for completing a task were adjusted on a daily basis. Iterations ended with documenting the iteration, presenting the new version of the metrology domain model to the supervisor and some stakeholders. During these meetings with the stakeholders, the input was gathered about the features for the next iteration.

In the beginning of the project, due to the lack of knowledge or the fear of uncertainties, the sum of estimates for all features in one sprint was rather big. As the time went by, the estimates stabilized to around 20 points (2 points = 1 working day). The development velocity chart (shown in Figure 15) also reflects periods in which there were some other non-project-related activities (e.g., leave days due to public holidays or job interviews).



Sprint implementation days				Effort	Remaining on implementation day…								
Trend calculated based on last	9	Days	Totals	66	66	57	37	31	27	19	13	8	
Task name	Story ID	Responsible	Status	Est.	1	2	3	4	5	6	7	8	9
	31	MLUN	Done	3	3	1	0	0	0	0	0	0	0
	31	MLUN	Done	3	3	1	0	0	0	0	0	0	0
	31	MLUN	Done	3	3	1	0	0	0	0	0	0	0
	35	MLUN	Done	8	8	8	4	0	0	0	0	0	0
	31	MLUN	Done	8	8	6	0	0	0	0	0	0	0
	35	MLUN	Done	2	2	1	0	0	0	0	0	0	0
	31	MLUN	Done	8	8	8	2	0	0	0	0	0	0
	34	MLUN	Done	4	4	4	4	4	4	0	0	0	0
	34	MLUN	Done	12	12	12	12	12	8	4	2	0	0
	34	MLUN	Done	4	4	4	4	4	4	4	0	0	0
	36	MLUN	Done	2	2	2	2	2	2	2	2	2	0
	36	MLUN	Done	2	2	2	2	2	2	2	2	2	0
	36	MLUN	Done	3	3	3	3	3	3	3	3	3	0
	35	MLUN	Done	4	4	4	4	4	4	4	4	1	0

Figure 14. Example work breakdown – 6th iteration

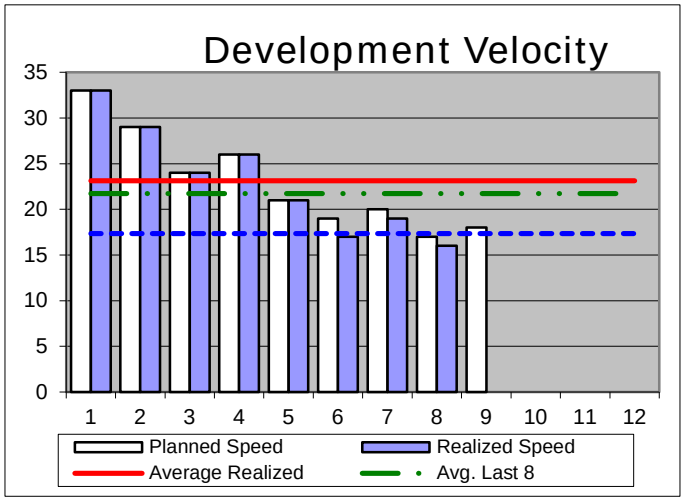


Figure 15. Development velocity per iteration

A list of all features was maintained in the product backlog, which contains plans for all iterations and phases. The features with the highest priority were tackled first. Every couple of iterations the backlog was replenished with new features and the iteration plan was adapted. Also, in the prototyping phase, the features that were capturing new domain concepts were prioritized over the 'nice-to-have' features.

The evolutionary approach was suitable for the domain model development. By frequently demonstrating the capabilities of the domain model, the stakeholders easily kept track of the progress and actively thought of new ways to use it. Furthermore, the model stimulated the metrologists to have a different point of view of the software and the way the sequences are designed.

8.Project Retrospective

8.1 Reflection

As described in Section 7.1, the project started with a learning phase where a lot of information about ASML and metrology had to be digested in a period of a couple of weeks. This period was also used to establish the scope of the project and the initial set of requirements. In this phase of the project, it was particularly important to ask the stakeholders to prioritize their requests: which requirements must be done and which ones would be nice to have, what functionality should be in the first release of

software and what in subsequent releases. If the stakeholders prioritize their requests, the overall project planning becomes easier.

Another thing that helped to keep project on track was arranging regular meetings with the domain experts. These meeting were used to demonstrate modeled domain concepts and to gather additional domain knowledge. The metrology domain experts are generally busy, thus they did not follow the progress of the project closely and the meetings were short. For that reason, it was crucial not to overload them with project details, but set the goals for each meeting and keep the discussions on a functional level (not diving into implementation details).

After the first three months, the most pleasant moment of surprise of the whole project occurred. During a meeting, after examining the configuration files for the domain objects, one of the domain experts asked: *"We've been talking about the functional design all this time… when do we talk about software design?"* At that point, the advantages of the domain-driven design were obvious: the configuration files, used directly in software, were viewed upon by domain experts as functional design. Traditionally, the functional design was always looked at as something separate from the software design, while this project was aiming to bring the two closer. During this meeting the functional metrologists (i.e., the domain experts) realized that the effort needed to translate their thoughts to a software implementation could be significantly reduced if the software would capture their way of thinking.

Sometimes the discussion in the meetings had to be steered in a more subtle way. There are people who have difficulties in discussing domain concepts in abstract terms and having even an incomplete or imperfect model proved to be useful in discussions with them. The domain concepts that were (intentionally) modeled incorrectly sparked discussions of they should look. Furthermore, the discussions also went into a direction of how things should look (as the current way of doing things was perhaps not the optimal one).

Closer to the end of the project, many questions were raised about how the metrology domain model can be used in the daily work of metrologists. In such discussions, it became apparent that structural changes (i.e., software redesign, architectural initiatives) cannot be done "from the outside," but from within the teams. A dedicated team of designers can only analyze software as a whole and establish reference architecture, but the actual redesign has to be done by the engineering teams. In other words, change can be initiated from the outside, but it has to be realized from within the engineering groups.

8.2 Design opportunities revisited

The design criteria that were selected in Section 2.5 as relevant for this project are: Genericity, Functionality, and Complexity.

Genericity regards the extent to which the designed artifact can be re-used, and the extent to which a best practice has been developed that can be applied in different situations. In this aspect, the metrology domain model provides a general framework for specifying domain objects through configuration files which are interpreted by the domain model itself.

In addition, all technology related aspects have been isolated from the domain model, so that it can be reused no matter what the underlying technologies are. Moreover, the model exposes generic interfaces where technology-dependent functionality can be plugged in. All of these are a consequence of applying domain-driven design and the architectural approach described in Section 4.1.

Functionality regards the extent to which the artifact satisfied the requirements. This criterion is reflected in the fact that the domain model had to capture all the data relations, constraints, and behavior of the metrology domain. An important distinc-

tion had to be made between the functionality that should be a part of the domain model and the functionality that should be implemented by the clients of the domain model. Additionally, a series of prototypes was developed to showcase how the domain model realizes all of its functional requirements.

Complexity design criterion, in particular the *Reduction of complexity* indicator, regards the hierarchical decomposition in the design, that is: the subdivision into compound components leading to a more transparent and more comprehensible design. The challenge during this project was to reduce the structural complexity of the software by creating a software component that is on a higher abstraction level. The domain concepts, such as sequences, measurements, or recoveries, were extracted by talking to domain experts and examining the current code base. As a result, the domain model can speed up the realization functional requirements in software while making the code base smaller.

Glossary

DDD Domain driven design. An approach to develop software for complex needs by connecting the implementation to an evolving model of the domain. 11

Metrology The science of measurement. 2

NoSQL Not only SQL. Database technology for storage and retrieval of data that uses looser consistency models than traditional relational databases. SQL stands for Structured Query Language. 8

PDEng Professional Doctorate in Engineering. Dutch degree awarded to graduates of engineering programs who develop their capabilities to work within a professional context. 5

Photolithography Process used in semiconductor device fabrication to transfer a pattern from a photomask (also called reticle) to the silicon surface of a substrate (also called wafer). Also referred to as optical lithography or UV lithography.....1

REST Representational State Transfer. An architectural style for distributed hypermedia systems....8

Bibliography

[1] ASML, "ASML: About ASML - About ASML," [Online]. Available: http://www.asml.com/asml/show.do?lang=EN&ctx=271. [Accessed 05 August 2013].

[2] BlackWasp, "The SOLID principles," [Online]. Available: http://www.blackwasp.co.uk/SOLIDPrinciples.aspx. [Accessed 05 August 2013].

[3] K. van Hee and K. van Overveld, "Criteria for assessing a technological design," 2010.

[4] D. Haughey, "MoSCoW Method," [Online]. Available: http://www.projectsmart.co.uk/moscow-method.html. [Accessed 05 August 2013].

[5] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2003.

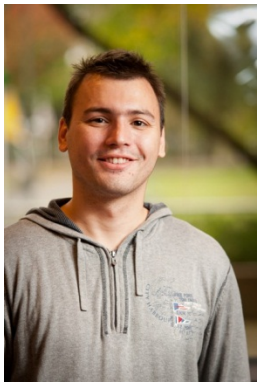
[6] J. Palermo, "The Onion Architecture," 29 July 2008. [Online]. Available: http://jeffreypalermo.com/blog/the-onion-architecture-part-1/. [Accessed 05 August 2013].

[7] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.

[8] V. Vernon, Implementing Domain-Driven Design, Addison-Wesley, 2013.

[9] M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley Professional, 2009.

About the Author



Matija Lukic received his M.Sc. degree in Computer Engineering and Information Theory in September 2011 from the School of Electrical Engineering, University of Belgrade. His master thesis was titled "Delay Monitoring Framework," which tackled the problem of latency-based performance analysis of the servers. He worked for a financial software vendor (Teletrader) where he developed an internal delay monitoring system that relied on his master thesis project. During master's studies, he spent one year as a member of the executive board of the international association of technical students called BEST (Board of European Students of Technology). After completing his master's studies, he immediately joined the Stan Ackermans Institute as a trainee in the Software Technology (ST) technological designer program. In January 2013, he joined ASML Netherlands B.V., Veldhoven, to conduct his nine-month final project in partial fulfillment of the requirements for the degree of Professional Doctorate in Engineering (PDEng) at the Stan Ackermans Institute.

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.