Analysis Toolset for Calibration, Performance and Diagnostic (CPD) Applications

Dongqi Hu
September 2016
Analysis Toolset for Calibration, Performance and Diagnostic (CPD) Applications

Eindhoven University of Technology
Stan Ackermans Institute / Software Technology

Partners

ASML Netherlands B.V. Eindhoven University of Technology

Steering Group
John Vugts (ASML)
Wouter Tabingh Suurmondt (ASML)
Sven Weber (ASML)
Kees van Berkel (TU/e)

Date
September 2016

Document status
Public

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct
Abstract

CPD applications are a group of test applications aimed to perform calibration, performance check and diagnostics on the machine in ASML. In order to gain insight in CPD applications, a toolset is needed to automatically process the data. This report describes the development of a configurable toolset that can get some selected properties of CPD applications. The toolset consists of 1) a database that stores the processed data, 2) a front-end that enables the user to analyze the CPD data by means of SQL queries, and 3) a converter that converts raw CPD data into a CPD-data model and stores it in the database. The toolset is designed to be extendible that the data of new properties can be obtained by adding new adapters, which are designed to deal with the diversity of data source. A prototype is implemented and several adapters are implemented to demonstrate the capability and extendibility of the toolset.

Keywords

Data Processing, Extendibility, CPD, ASML, SAI, Software Technology, OOTI
Foreword

'It does not matter how slowly you go so long as you do not stop.' (Conficius)

Dear Dongqi,

After many months, your project is almost finished. The vague ideas about CATS, as we had in mind, have been transformed into a real solution that can be experienced by anyone. This all is the result of your hard work. And although ASML is a slow adapter of new ideas, I am convinced that CATS (CPD Analysis ToolSet) fulfills a need and will be soon adopted and recognized for its value. And this is in many cases the issue with new ideas: they will be valued afterwards.

But even more import, this process also reflects your personal learning. I remember that, when you started, you expected to get instructions and only have to build the solution. However, you soon found out that the approach was completely different. There was almost nothing and you have to find everything by yourself. For example, the stakeholders, the users and the requirements. You also had to learn and understand the non-standard ASML development environment. And finally, you had to understand ASML’s issues and concerns with respect to interfacing of software components.

It was a steep learning trajectory. You probably learned more about organizing a project, sharing your ideas and selling the solution, rather than building it. But this is exactly the purpose of such graduation project. We all know that you are a good engineer, but the PDEng education expects you also to manage your own project. We had many, many discussions about this point and also your supervisor Kees van Berkel gave you lots of instructions how to organize your work.

However, it all pays off. I see a big change in your professional attitude from a designer towards an architect, from someone who just built the system into someone who understands the customer needs and invents a solution. You are definitely on the right track. As Confucius already stated ‘It does not matter how slowly you go so long as you do not stop.’ Consequently, continuation of learning for the rest of your life is the only option.

Thanks a lot for your work, it was a great pleasure learning and cooperating with you. I wish you all the best in your future life,

Wouter Tabingh Suermondt,

Veldhoven, June 2016
Preface

This report summarizes the project “Analysis Toolset for Calibration, Performance and Diagnostic (CPD) Applications” executed by Dongqi Hu as his graduation project for the OOTI PDEng (Professional Doctorate in Engineering) program provided by the Department of Mathematics and Computer Science of Eindhoven University of Technology and Stan Ackermans Institute. It is a nine months project executed in ASML under the supervision of Prof. Kees van Berkel and Wouter Tabingh Suermondt.

The goal of this project is to develop a toolset that is able to automatically get some properties of CPD applications, which is a group of test applications aimed to perform calibration, performance measurement and diagnostics on the machine. Based on the data obtained by the toolset, some analyses can be performed to gain the insight of the CPD applications.

This report is primarily intended for readers with software engineering background, especially the people who are interested in big data processing and developing software in a complex domain.

Dongqi Hu
August 2016
Acknowledgements

First, I would like to gratefully and sincerely thank my company supervisor Wouter Tabingh Suermontd for his tutoring and supporting all through the project. He is a knowledgeable architect with rich experience. I learned a lot from him.

I want to acknowledge the help from my university supervisor Prof. Kees van Berkel. I would like to thank him for his help in managing the process of the project and his valuable feedbacks.

Besides the supervisors, many people inside ASML gave me lots of help during the project. They are: Marcel van Heerebeek, Danny Handoko, Guus Weemaes, Ling Ji, Bram Schoenmakers, Ben Vandeveld, Jerry van Kampen, Teun van de Berg, Sander van Woensel, Paul Slits, John Vugts, Sven Weber, Christian Spronk, Alexey Nesterenko and Walfried Veldman. I appreciate their help very much.

OOTI program gives continuous support to its trainee during the program. Thank Judy Strother for the technical writing workshop and the review of the report. Thank the coaches (Peter Zomer, Cynthia Schreuder, Marloes Hartman and Sandra van Dongen) for their help in personal and professional development. Thank Desiree for organizing everything. Thank all my colleagues for the help during the program. Finally, Thank Ad Aerts, the program director.

Thank my family and my friends for their support during my stay in the Netherlands.

Dongqi Hu
August 2016
Executive Summary

CPD applications are a group of test applications in ASML aimed to perform calibration, diagnostics and performance measurement on the machine. The development of a CPD application usually involves both functional engineer and software engineer and many communications and checks are needed. The reuse in CPD applications is low and the maintenance is expensive and time consuming.

In order to optimize the development process of CPD applications, a new toolkit CIDT (CPD Integrated Development Toolkit) is under development. CIDT is aimed to be used by functional engineers and generate CPD applications automatically. To develop CIDT, the fundamental insight and interface stability status in CPD applications need to be obtained by processing and analyzing the data of CPD applications. However, ASML has developed about 1500 unique CPD applications and around 1000 of them are still in use today. It is a huge workload to manually process the data of CPD applications. A better solution is to develop a toolset to process the data automatically.

This project describes the development of a toolset to get the properties of CPD applications. In the problem analysis, the design constraints and design options are discussed. An initial list of requirements is collected through meetings with stakeholders. Architecture is designed based on the designed constraints and the requirements. Finally a prototype is implemented and several adapters are implemented to demonstrate the extendibility and capability of the toolset.

The architecture of the toolset contains three elements: the database, the frontend and the converter. The database is used to store the processed data. The frontend is the graphical user interface through which the user can operate the toolset. The converter can be used to process the data in the data source and store the processed data into database. The three elements are decoupled from each other by interfaces and can be updated or extended without heavily affecting the others.

The adapter is designed to deal with the different kinds of data source. It brings benefits in the following:

- It decouples the framework of the toolset from the data source.
- It makes the toolset extendible. A new property of CPD applications can be obtained by adding a new adapter to the toolset.

The toolset brings benefits to the stakeholders by processing the data of CPD applications automatically. Based on the data obtained by the toolset, some analysis can be performed. It is recommended that the toolset be extended with more adapters.
# Table of Contents

Foreword ........................................................................................................... i
Preface ............................................................................................................... iii
Acknowledgements ......................................................................................... v
Executive Summary ....................................................................................... vii
Table of Contents ........................................................................................... ix
List of Figures ................................................................................................. xiii
List of Tables .................................................................................................... xv

1. Introduction ................................................................................................. 1
   1.1 Context .................................................................................................. 1
   1.2 Outline .................................................................................................. 3

2. Stakeholder Analysis ................................................................................... 5
   2.1 Introduction .......................................................................................... 5
   2.2 ASML .................................................................................................... 5
   2.3 Eindhoven University of Technology (TU/e) ....................................... 6
   2.4 Analysis ............................................................................................... 6

3. Problem Analysis ......................................................................................... 9
   3.1 Context ............................................................................................... 9
   3.2 Requirements Analysis ........................................................................ 11
   3.3 Design Constraints ............................................................................ 11
   3.4 Design Options .................................................................................. 11
   3.5 Design Opportunities ........................................................................ 13

4. Domain Analysis ......................................................................................... 15
   4.1 Introduction ........................................................................................ 15
   4.2 CPD Application ............................................................................... 15
   4.3 Interface .............................................................................................. 17
   4.4 Software Release ................................................................................ 19
   4.5 Data Model ......................................................................................... 20

5. Feasibility Analysis ..................................................................................... 22
   5.1 Issues ................................................................................................ 22
   5.1.1 Format Change of Data Sources .................................................... 22
   5.1.2 Separate ADT Applications from CPD Applications ................. 22
List of Figures

Figure 1 Development Process of CPD Application .............................................. 1
Figure 2 Development Process of CPD Application by Using CIDT .................... 2
Figure 3 Stakeholder Analysis by Interest and Power ......................................... 7
Figure 4 Fundamental Blocks of CPD Applications .......................................... 16
Figure 5 How TWINSCAN Software Is Organized ........................................... 18
Figure 6 Split Off of QBL .................................................................................. 19
Figure 7 Data Model ......................................................................................... 20
Figure 8 Illustration of the Design That Treating Data Sources Independently ......... 22
Figure 9 Context Diagram ............................................................................... 25
Figure 10 Use Case ........................................................................................ 26
Figure 11 Top Level Architecture .................................................................... 29
Figure 12 Architecture of Converter ................................................................. 30
Figure 13 Data Flow of Converter .................................................................... 31
Figure 14 Architecture of Frontend .................................................................. 31
Figure 15 Class Diagram of Converter ............................................................... 34
Figure 16 Sequence Diagram of Converter ...................................................... 35
Figure 17 Design Model of Internal Database .................................................. 36
Figure 18 Mockup of the User Interface ............................................................ 36
Figure 19 Class Diagram of the Frontend .......................................................... 37
Figure 20 State Machine of Converter in the Frontend ..................................... 38
Figure 21 Class Diagram of Database Management .......................................... 38
Figure 22 Sequence Diagram of Query Management in the Frontend .......... 39
Figure 23 Sequence Diagram of Converter Management in the Frontend .... 39
Figure 24 Sequence Diagram of Database Management in the Frontend ...... 40
Figure 25 SQL Statement of Database Creation .............................................. 42
Figure 26 User Interface .................................................................................. 43
Figure 27 Output Example of Self-Defined Functions ........................................ 43
Figure 28 Context Menu of Folder Item .............................................................. 47
Figure 29 Context Menu of Query Item ............................................................... 48
Figure 30 Converter Menus ............................................................................ 49
Figure 31 Converter Configuration Items .......................................................... 49
Figure 32 Converting Status of Converter ....................................................... 49
Figure 33 Stopped Status of Converter .............................................................. 50
Figure 34 Menus of Database Management .................................................... 50
Figure 35 Message Box of 'Flush DB' Submenu ............................................. 50
Figure 36 Dialog Box of ‘Show Info’ Submenu ................................................. 50
Figure 37 Work-Breakdown Structure of This Project ...................................... 57
Figure 38 Gantt Chart ..................................................................................... 58
List of Tables

Table 1 Main Stakeholders in ASML ........................................5
Table 2 Main Stakeholders in TU/e ........................................6
Table 3 Stakeholder Analysis by Interest and Power .................6
Table 4 Possible Data Sources of CPD Applications ...............9
Table 5 Content of Interface Viewer ....................................10
Table 6 Content of Scope File ............................................10
Table 7 Metrics of TICS [2] ...............................................10
Table 8 Comparison of With and Without Internal Database ....12
Table 9 Static Properties of CPD Applications .......................16
Table 10 Life-Cycle Properties of CPD Applications ..............16
Table 11 Run-Time Properties of CPD Applications...............16
Table 12 Functional Requirements for Query Management ......27
Table 13 Functional Requirements for Database Management ....27
Table 14 Functional Requirements for Converter Management ...27
Table 15 Domain Requirements .........................................28
Table 16 State Transition Table .........................................38
Table 17 Implementation of Adapters ..................................41
Table 18 Feedback from the Stakeholder ..............................45
Table 19 Milestones .......................................................59
Table 20 Glossary ..........................................................63
1. Introduction

This chapter describes the context and the goal of the project. The outline section gives a brief introduction of what is discussed in the following chapters.

1.1 Context

ASML lithography scanners are designed to deliver very high throughput with nanometer accuracy. To achieve this extreme performance, a huge number of system parameters have to be calibrated and the performance has to be measured and analyzed. CPD application, which is short for calibration, performance and diagnostic application, has been and is being developed to complement these basic functionalities.

The development process of CPD application is shown in Figure 1. This process mainly contains the following steps [1]:

1. Functional engineers describe the functional specification.
2. Together with software engineers, they describe the software specification based on the functional specification.
3. Software engineers develop the code.
4. Software engineers test the code.
5. Functional engineers validate the functionality of the code.

![Figure 1 Development Process of CPD Application](image)

The main problems of this process are:

- Development time is long.
  Because this process involves engineers from two different domains (functional engineers and software engineers), the communication and knowledge translation usually consume lots of time.
  The code may not pass the validation because of mismatch with functional specifications.
- Maintenance is expensive.
  If there is an update on a CPD application, all the documents (including the functional specifications, software specifications, test documents and validation documents) need to be updated.
  CPD applications share similarities in term of structure and types of actions and measurements. However, due to historical reasons, the reuse is low.
To solve these problems, a CPD integrated development toolkit (CIDT) project was started. Figure 2 shows the development process of CPD applications by using CIDT. In Figure 2 there is no software engineer. Only functional engineers are involved in the development process. In other words, the functional engineer can develop software by himself without knowledge of software design details. The main steps of developing a new CPD application by using CIDT are:

1. Functional engineers describe the functional specification.
2. Functional engineers use models to design the CPD application.
3. Functional engineers simulate behavior of the models to verify its functionality.
4. CIDT uses models as input and generate the code of the CPD application automatically.
5. Functional engineers validate the functionality of the CPD application.

![Development Process of CPD Application by Using CIDT](image)

Following are some benefits of developing CPD applications by using CIDT:

- Developing a new CPD would be much faster. There is no knowledge translation between functional engineers and software engineers. Functional engineers are at the center of development.
- Maintenance would be much easier. The content of maintenance is shifted from CPD applications and documents to models and CIDT. Many model blocks could be reused if the models are well designed.
- The functionality of CPD applications can be verified by simulation of the models.

In order to develop CIDT, substantial knowledge of CPD applications is needed. Some of the knowledge are listed below:

- Required interfaces
- Historical evolution of required interfaces
- Volumes of CPD applications per functional cluster

A toolset is needed to get such knowledge automatically. ASML has developed about 1500 unique CPD applications in the past 15 years and roughly 1000 of those are still in use today. Moreover, most of them are still updated to varying degrees and with varying frequency. It is a huge workload to manually process the data of CPD applications.
This project is about the design and development of a toolset to process the data of CPD applications. It focuses on CPD applications in TWINSCAN platform. It aims to support the development of CIDT by the followings:

- Providing a toolset that is able to process data of CPD applications;
- Analyzing the interface stability status of CPD applications based on the data obtained by the toolset.

### 1.2 Outline

In the next four chapters, some analyses are described. In Chapter 2, the stakeholders involved in this project and their interests are analyzed. In Chapter 3, the problem analysis is performed. Some high level requirements are analyzed, followed by the analysis of the design constrains and options. Finally, the design opportunities are described. In Chapter 4, the key elements in the domain are analyzed and a data model is created based on the relationships between the elements. In Chapter 5, the issues and possible risks in this project are analyzed.

In the chapters from Chapter 6 to Chapter 10, the development of the toolset is described from requirements to verification and validation.

In Chapter 11, it is described how to use and extend the toolset.

In the following three chapters, the report is wrapped up with conclusions and reflections. The project management is also described.

Note: In order to protect the Intellectual Property (IP), some data and names (for example, component names) used in this report are faked.
The previous chapter introduces the context of this project. This chapter analyzes the stakeholders involved in this project and their interests.

2.1 Introduction
In this project, there are two main organizations involved: ASML and Eindhoven University of Technology. The following describes each organization and the stakeholders in it.

2.2 ASML
ASML is the provider of this project. The initiator, owner and manager of this project are all employees of ASML. They are important stakeholders as individual persons.

The CIDT project team is responsible for investigating and developing CIDT, which aims to simplify and standardize the development of CPD applications. As a team, it is an important stakeholder of this project. The initiative of this project is to provide support to CIDT.

CPD competence team is another important stakeholder as a team. It works on standardizing and optimizing the development of CPD applications. Its missions include:
- Facilitating and providing training to CPD developers
- Developing all means and methods as required
- Facilitating and controlling guidelines, designing rules and templates

These stakeholders have several interests in this project.

The first interest is to know the external interface stability of CPD applications. This is very important to the development of CIDT. If one interface is changed, the CPD applications that require that interface should also be changed. However, CIDT is used by functional engineers and the CPD applications generated by CIDT are in a separate software archive. The change of interfaces would result in the failure execution of CPD applications. The more an interface changes, the more a CPD application failures. Functional engineers would be very unsatisfied with CIDT if CPD applications fail very often, which is the result of interface instability. So interface stability is a key point to the success of CIDT.

The second interest is to trace the development and update of CPD applications. By tracing the CPD applications, it can be monitored that whether CPD applications are developed in the correct/standard way.

As an addition, cooperating with the university to import new knowledge/technology/process is also an interesting factor to stakeholders in ASML.

Table 1 shows the main stakeholders in ASML.

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Position</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wouter Tabingh Suermontd</td>
<td>System Architect of CIDT</td>
<td>Project mentor; company supervisor</td>
</tr>
<tr>
<td>Marcel Van Heerebeek</td>
<td>Leader of CPD Competence Team</td>
<td>Technical mentor</td>
</tr>
<tr>
<td>Sven Weber</td>
<td>System Architect Software</td>
<td>Project manager</td>
</tr>
<tr>
<td>John Vugts</td>
<td>ESD Department</td>
<td>Project owner</td>
</tr>
</tbody>
</table>
2.3 Eindhoven University of Technology (TU/e)

The Professional Doctorate in Engineering (PDEng) degree program in software technology (OOTI) is provided by the Department of Mathematics and Computer Science of Eindhoven University of Technology. This project is a part of the trainee’s education. TU/e is concerned with the design aspects and project management of this project.

The stakeholder from the university is Prof. Kees van Berkel. He is a full professor of Security and Embedded Networked Systems at the department of Mathematics and Computer Science of TU/e. As the university supervisor, his role includes:

- Monitoring and controlling the quality and progress of the project and the resulting products
- Reviewing the project report with respect to the technical and academic contents

Another stakeholder from the university is Dongqi Hu, the PDEng trainee. His responsibility is to design and implement the product in a well-designed process. His main interest is to graduate with PDEng degree.

Table 2 shows the main stakeholders in TU/e.

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof. Kees van Berkel</td>
<td>University Supervisor</td>
</tr>
<tr>
<td>Dongqi Hu</td>
<td>PDEng Trainee</td>
</tr>
</tbody>
</table>

2.4 Analysis

Stakeholders are analyzed by their interests and powers in this project, as shown in Table 3 and Figure 3.

Wouter Tabingh Suermondt, Prof. Kees van Berkel and Dongqi are the key players in this project. They are involved in every detail of this project. Dongqi Hu is the PDEng trainee who is trying to prove his design ability through this project. Wouter Tabingh Suermondt is the company supervisor who provides help in the company. He is also the most important client of the toolset that is going to be developed. Prof. Kees van Berkel is the university supervisor who monitors and controls the progress.

Sven Weber and John Vugts are concerned with the overall progress of the project and do not have much interest in the details. They should be notified of the project progress regularly.

Marcel van Heerebeek is the team leader of the CPD competence team. He is interested in the development of the toolset which could possibly bring benefit to the CPD competence team. He is also an expert of CPD development who can provide both knowledge and requirement input and feedback. The CPD competence team should be involved and consulted in the development process of this project.

Table 3 Stakeholder Analysis by Interest and Power

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>( W_{\text{Interest}} )</th>
<th>Interest</th>
<th>( W_{\text{Power}} )</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wouter Tabingh Suermondt</td>
<td>8</td>
<td>High</td>
<td>9</td>
<td>High</td>
</tr>
<tr>
<td>Marcel Van Heerebeek</td>
<td>7</td>
<td>High</td>
<td>4</td>
<td>Low</td>
</tr>
<tr>
<td>Sven Weber</td>
<td>4</td>
<td>Low</td>
<td>6</td>
<td>High</td>
</tr>
<tr>
<td>Name</td>
<td>Interest</td>
<td>Power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>John Vugts</td>
<td>Low</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prof. Kees van Berkel</td>
<td>High</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongqi Hu</td>
<td>High</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3 Stakeholder Analysis by Interest and Power
3. Problem Analysis

In the previous chapters, the context of this project and the stakeholder analysis are described. This chapter describes the problem analysis. Some high level requirements are analyzed, followed by the analysis of the design constraints and options. Finally, the design opportunities are described.

3.1 Context

As mentioned in Chapter 1, this project is to develop a toolset that is able to process the data of CPD applications. With the data obtained by the toolset, the stability of external interfaces required by CPD applications could be analyzed.

In order to develop such a toolset, the first question would be: where are the data sources of CPD applications? With several weeks’ study on the code archive, databases, and some tooling in the internal network of ASML, some possible data sources of CPD applications were collected, as shown in Table 4. For some terminologies in Table 4 to Table 7, please refer to Table 20 Glossary.

Table 4 Possible Data Sources of CPD Applications

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Format</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Viewer</td>
<td>CSV File</td>
<td>Information related to interface. For details, see Table 5 Content of Interface Viewer</td>
</tr>
<tr>
<td>Scope File</td>
<td>Text File</td>
<td>Static software architecture information. For details, see Table 6 Content of Scope File</td>
</tr>
<tr>
<td>XPTM File</td>
<td>Text File</td>
<td>Information of test applications, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Application description</td>
</tr>
<tr>
<td>PKG_contents File</td>
<td>Text File</td>
<td>Files that need to be installed on the machine are specified. Information includes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Target filename on the machine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The package name the file belongs to.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The source filename in the development environment.</td>
</tr>
<tr>
<td>Makefile</td>
<td>Text File</td>
<td>Information of how to compile and link an application</td>
</tr>
<tr>
<td>Scout</td>
<td>File</td>
<td>Symbol usage by other elements, including:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Interface and interface level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Symbol and symbol level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Provider, client and level of provider-client relation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Client RP</td>
</tr>
<tr>
<td>TICS</td>
<td>Database</td>
<td>Static code analysis. For details of code quality metric, see Table 7 Metrics of TICS</td>
</tr>
<tr>
<td>Clearquest</td>
<td>Database</td>
<td>Architecture information, including AS, RP, FC, BB, CC, IF, layer and module. Software changes, history actions and time. Basic information of product, platform, project, release. Work instruction.</td>
</tr>
<tr>
<td>Source Code</td>
<td>Text File</td>
<td>The source codes of CPD applications</td>
</tr>
<tr>
<td>AIR</td>
<td>Database</td>
<td>External changes for D&amp;E</td>
</tr>
<tr>
<td>IP</td>
<td>Database</td>
<td>External changes</td>
</tr>
<tr>
<td>5 Time Why</td>
<td>File</td>
<td>The root cause of errors</td>
</tr>
</tbody>
</table>
Table 5 Content of Interface Viewer

<table>
<thead>
<tr>
<th>Content</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure information</td>
<td>• Assembly, RP, FC, BB, CC, Element, DO’s</td>
</tr>
<tr>
<td></td>
<td>• Level, Level details</td>
</tr>
<tr>
<td></td>
<td>• Layer</td>
</tr>
<tr>
<td></td>
<td>• Scope file</td>
</tr>
<tr>
<td>IMAS metric</td>
<td>• Last year change frequency</td>
</tr>
<tr>
<td></td>
<td>• Build dependencies</td>
</tr>
<tr>
<td></td>
<td>• Directly dealt interface</td>
</tr>
<tr>
<td></td>
<td>• Indirectly dealt interface</td>
</tr>
<tr>
<td></td>
<td>• Encapsulation</td>
</tr>
<tr>
<td></td>
<td>• IMAS compliancy</td>
</tr>
<tr>
<td></td>
<td>• Build scope</td>
</tr>
<tr>
<td>Details on the dependencies of assembly</td>
<td>• File with the most dependencies</td>
</tr>
<tr>
<td></td>
<td>• Dependencies per file type</td>
</tr>
<tr>
<td>Information based on scope sets</td>
<td>• Used by external Applications</td>
</tr>
<tr>
<td></td>
<td>• Used by components of other assemblies, RP’s, FC’s, BB’s</td>
</tr>
<tr>
<td></td>
<td>• Direct buildscope components</td>
</tr>
<tr>
<td></td>
<td>• Excess buildscope components</td>
</tr>
<tr>
<td></td>
<td>• Direct dealt by system I/F</td>
</tr>
<tr>
<td></td>
<td>• Direct dealt by I/F of other assemblies</td>
</tr>
<tr>
<td></td>
<td>• Indirect dealt by I/F of other RP’s, FC’s, BB’s</td>
</tr>
<tr>
<td></td>
<td>• Direct dealt I/F’s by this interface</td>
</tr>
<tr>
<td></td>
<td>• Indirect dealt I/F’s by this interface</td>
</tr>
<tr>
<td>Based on the CSScout symbol usage analysis</td>
<td>Percentage of the symbols used on level: assembly, RP, FC, BB, CC</td>
</tr>
<tr>
<td>Interface symbols used by other interfaces.</td>
<td>Interface symbols used by other interfaces in levels: other assembly, own assembly, other RP, own RP, other FC, own FC, other BB, own BB.</td>
</tr>
<tr>
<td>Interface symbols used by other elements.</td>
<td>Interface symbols used by other elements in levels: assembly, RP, FC, BB and CC</td>
</tr>
</tbody>
</table>

Table 6 Content of Scope File

<table>
<thead>
<tr>
<th>Content</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal building-block level scope files</td>
<td>• BB name, 11Inc, contains, depends_on</td>
</tr>
<tr>
<td></td>
<td>• CC layer, requires</td>
</tr>
<tr>
<td></td>
<td>• IF contains, requires</td>
</tr>
<tr>
<td>External building-block level scope files</td>
<td>• BB depends_on</td>
</tr>
<tr>
<td></td>
<td>• IF contains, requires, provided_at</td>
</tr>
<tr>
<td>FC level scope files</td>
<td>• FC name, contains, depends_on</td>
</tr>
<tr>
<td>Assembly level scope files</td>
<td>• AS name, contains, depends_on</td>
</tr>
</tbody>
</table>

Table 7 Metrics of TICS [2]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loc</td>
<td>Lines of code</td>
</tr>
<tr>
<td>Eloc</td>
<td>Effective lines of Code</td>
</tr>
<tr>
<td>Gloc</td>
<td>Lines of code including generate code</td>
</tr>
</tbody>
</table>
3.2 Requirements Analysis
Through discussion with the company supervisor, some important requirements of the toolset are listed:

- A graphical user interface should be provided, through which the user can interact with the toolset.
- Through GUI, the user can flush the internal database.
- Through GUI, the user can check metadata of the internal database.
- The toolset should always show a response to the operations.
- The toolset can perform analysis without user intervention. One scenario is running the toolset during night.

3.3 Design Constraints
The design constraints in this project are listed:

1. Because of security issue, it is not allowed to run a server inside ASML without a strong reason. After discussion with company supervisor, it is agreed that the toolset would be developed as a standalone application.

2. Software archive is strictly only available within the Software development environment via Linux VDI:
   - No direct interfaces to ClearCase via local PC
   - No indirect access to ClearCase via SSH or other remote protocol’s
As a result, in this project, the toolset is only able to be used in the Linux VDI. Otherwise, it cannot access the ClearCase software environment.

3. In the Linux VDI, developers cannot freely install whatever software they want. They can only use what is available. As a result, in the development of the toolset, available software/libraries are given priority to be used.

3.4 Design Options
One option in this project is whether to use an internal database or not. An internal database could be used in the toolset to store processed data and be queried by the user to retrieve the data. The toolset can also be developed without the internal database. In order to decide whether to use an internal database, Table 8 shows a comparison of the two ways of design from technical perspective.

### Table 8 Comparison of With and Without Internal Database

<table>
<thead>
<tr>
<th></th>
<th>Without Internal Database</th>
<th>With Internal Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access all raw data in runtime</td>
<td>Yes.</td>
<td>Yes.</td>
</tr>
<tr>
<td></td>
<td>All the database access interfaces can be integrated into the toolset and the toolset is able to access all the data in the runtime</td>
<td>All the raw data can be accessed and transformed into the internal database.</td>
</tr>
<tr>
<td>Processed data reusability</td>
<td>No.</td>
<td>Yes.</td>
</tr>
<tr>
<td></td>
<td>No temporary cache to store the processed data.</td>
<td>All Processed data are stored in the internal database and can be reused</td>
</tr>
<tr>
<td>Small size</td>
<td>Yes.</td>
<td>No.</td>
</tr>
<tr>
<td></td>
<td>With no internal data stored.</td>
<td>Processed data are stored inside the toolset.</td>
</tr>
<tr>
<td>Customizability</td>
<td>Yes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Toolset is developed from scratch, so it can be designed according to the requirements of clients.</td>
<td></td>
</tr>
<tr>
<td>Extendibility</td>
<td>Yes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Toolset is developed from scratch, so it can be designed to be extendable.</td>
<td></td>
</tr>
<tr>
<td>Independent from the raw data</td>
<td>No.</td>
<td>Partly.</td>
</tr>
<tr>
<td></td>
<td>The raw data is the only data source.</td>
<td>The toolset can continue to use the processed data even though the access method or format of raw data is changed.</td>
</tr>
<tr>
<td>Fast Running speed</td>
<td>No.</td>
<td>Partly.</td>
</tr>
<tr>
<td></td>
<td>The toolset needs to access the raw data and process it in runtime. It takes some time (from minutes to days) for the user to get the result.</td>
<td>Converting the data from raw data source to internal database takes some time (from minutes to days). Query the data in the internal database.</td>
</tr>
</tbody>
</table>
If a point is assigned to each aspect as: Yes=1, partly=0.5, No=0, and each aspect has the same weight as 1, then we can get the total points of each way:

<table>
<thead>
<tr>
<th></th>
<th>Without Internal Database</th>
<th>With Internal Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total points</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Based on the result of the comparison, an internal database is used.

### 3.5 Design Opportunities

Three design criteria that would be applied in this project:
- **Extendibility:** this project is the initial step to develop a toolset to perform analysis on CPD applications. More features would be added later on. So extendibility should be taken into consideration.

- **Self-Descriptiveness:** the toolset would be maintained and extended by people who are not its developer. To make it easy to be maintained and extended, good readability is very important.

- **Separation:** it describes the degree to which a distinct piece of functionality is placed into a distinct component that has a well-defined interface to the rest of the world. In this project, separation of functionality means good extendibility and maintainability.

Two design criteria that would not be applied in this project:
- **Attractiveness:** the toolset is not a commercial tool and would be used only inside the company by technical people.

- **Maturity:** in this project, the toolset is more a proof of concept than a mature product.
4. Domain Analysis

“*A system that does not solve the business needs is of no use to anyone, no matter how pretty it looks or how well architected its infrastructure*”

--- Dan Haywood

4.1 Introduction

As described in Chapter 2, the main interests of stakeholders in ASML are interface stability and development and modification of CPD applications. CPD application and interface are the two main elements in the domain of this project and are analyzed in the following sections.

ASML has multiple products with different types. Correspondingly, it has multiple software releases for different products. Section 4.4 gives a simple introduction of software release in ASML.

At the end of this chapter, a data model is introduced to organize all data in this domain into one data structure. With this standardized data structure, it is much easier to process the data.

4.2 CPD Application

There are multiple definitions of CPD application:

- Calibration, performance and diagnostic applications (CPD applications) is a typical ASML terminology for a set of applications that are used to calibrate (part of) a subsystem, to test its behavior, to track malfunctioning electronics or mechanics. [3]

- CPD application is an abbreviation for Calibration, Performance and Diagnostic application. These applications are used to calibrate, diagnose or measure the performance of a machine during build-up, factory acceptance test, site acceptance test or planned maintenance. The abbreviations CPD, C&P and CnP all mean the same. On the old system, this was called "test software". [4]

- Test Software used during Calibration, Performance, and Diagnostic. This definition is from the ASML terminology database.

CPD applications are executables that are started by Test Manager. Test Manager is the application that controls all CPD applications. It allows the operator to select a certain CPD application from a list of supported CPD applications for a specific machine configuration and to start it. The Test Manager also supports defining and executing test queues. A test queue is a sequence of CPD applications that can be executed sequentially without user intervention (i.e., during the night or when a CPD application takes a very long time to execute). Not all CPD applications can be part of a test queue because some are interactive (i.e., for adjustment of a hardware component). [3]

There are four fundamental blocks that are present in every CPD application [6], as shown in Figure 4:

1. Definition of the inputs
2. Execution of the test, collect data
3. Processing of the collected data
4. Generate a report of the result

Sometimes steps 2 and 3 may be repeated a number of times.
Figure 4 Fundamental Blocks of CPD Applications

From the user’s point of view, five categories of CPD applications can be distinguished [3]:

- **Calibration**: calculate and adapt one or more machine constants.
- **Performance**: measure the functionality behavior of hardware. Usually some numbers are checked to test whether they are with a certain range.
- **Diagnostics**: locate machine problems.
- **Analysis**: processing and presenting data, gathered from other tests. These tests can run without hardware being present, only a simulate system is needed.
- **Adjustment**: interactive CPDs used to allow a mechanic to manually adjust a hardware function. There is a direct feedback because the test shows the information from the hardware in real-time on the console.

Through meetings with CPD domain experts, some properties of CPD applications are collected. They are split into three groups: static properties, life-cycle properties, and run-time properties are shown in Table 9, Table 10 and Table 11. For some terminologies in these tables, please refer to Table 20 Glossary.

**Table 9 Static Properties of CPD Applications**

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Data Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPD type</td>
<td>enumeration</td>
<td>calibration, performance, diagnostic, analysis, adjustment, other</td>
</tr>
<tr>
<td>FC</td>
<td>string</td>
<td>Not applicable</td>
</tr>
<tr>
<td>BB</td>
<td>string</td>
<td>Not applicable</td>
</tr>
<tr>
<td>required interface</td>
<td>string list</td>
<td>Not applicable</td>
</tr>
<tr>
<td>human interaction</td>
<td>Boolean</td>
<td>True, False</td>
</tr>
<tr>
<td>scenario support</td>
<td>Boolean</td>
<td>True, False</td>
</tr>
<tr>
<td>sequence runner support</td>
<td>Boolean</td>
<td>True, False</td>
</tr>
<tr>
<td>target machine type</td>
<td>string</td>
<td>Not applicable</td>
</tr>
<tr>
<td>framework used</td>
<td>string</td>
<td>Not applicable</td>
</tr>
<tr>
<td>number of steps</td>
<td>integer</td>
<td>&gt;=0</td>
</tr>
<tr>
<td>number of UI elements</td>
<td>integer</td>
<td>&gt;=0</td>
</tr>
</tbody>
</table>

**Table 10 Life-Cycle Properties of CPD Applications**

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Data Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>created release</td>
<td>string</td>
<td>Not applicable</td>
</tr>
<tr>
<td>modified release(s)</td>
<td>string list</td>
<td>Not applicable</td>
</tr>
<tr>
<td>change times of required</td>
<td>integer</td>
<td>&gt;=0</td>
</tr>
<tr>
<td>interface per year</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 11 Run-Time Properties of CPD Applications**

<table>
<thead>
<tr>
<th>Property</th>
<th>Data Type</th>
<th>Data Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time duration</td>
<td>Float</td>
<td>&gt;=0</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>selected scenario</td>
<td>String</td>
<td>Not applicable</td>
</tr>
<tr>
<td>results</td>
<td>Enumeration</td>
<td>ERROR, FAILED, FINISHED, OK</td>
</tr>
<tr>
<td>Measurement Quality</td>
<td>Enumeration</td>
<td>OK, NOK, Within Limit, Without Limit</td>
</tr>
</tbody>
</table>

Apart from CPD applications, there is another group of test software called ADT. The full name is advanced diagnostics tool. It is aimed to be used by expert users to perform low level control of the machine to diagnose it. About the relationship between ADT applications and CPD applications, there are two opinions in ASML:

1. ADT applications are a special kind of CPD applications. They are both test software to test the machine functionality.
2. ADT applications do not belong to CPD applications. They are different in the following aspects:
   - CPD applications are aimed for non-expert users, while ADT applications are aimed for expert users.
   - Most CPD applications could be part of a test queue and execute sequentially, while ADT applications do not support this, because ADT applications are applications interactive with the user.
   - As shown in Figure 4, CPD applications generate a report at the end of execution; while ADT applications do not generate a report.
   - On one machine, only one CPD can be executed at a certain time, while multiple ADT applications can be executed in parallel on one machine.

In this project, ADT applications are treated as different applications from CPD applications based on the differences described above.

### 4.3 Interface

Figure 5 shows how the TWINSCAN software is organized. First, some terminologies in Figure 5 are explained:

- Functional Cluster: group of software/hardware with the same system function
- Release Part: group of software for the same machine platform
- Assembly: group of software with the same lifecycle (branches, baselines)
- Layer: group of software with the same software responsibility
- Building Block: unit of configuration management and baselining
- Component: Atomic unit of development

Organizationally, TWINSCAN software is divided into functional clusters. Each functional cluster contains one or more building blocks and each building block contains one or more components. One building block belongs to exactly one functional cluster and one component belongs to exactly one building block.

Component is the basic unit for software development. It can serve different purposes [3]:

- Correspond with a specific part of the machine, such as a wafer stage or reticle handler
- Serve as common utility, such as communication network or error handler
- Co-ordinate activities, such as lot production
In the software architecture user manual, interface is described as following:

A component can provide interfaces and a component can require interface(s). A component can only be accessed via (one of) the component’s external interface(s), i.e., the interface that it provides/implements for other components. An interface is a named set of syntax/API, behavior and performance. An interface is self-contained and preferably contains no references to interfaces defined by other components. Sometimes this is hard to avoid, for example when passing parameters to lower-level functions. In case of common datatypes interfaces have to reference the other interfaces.[3]

Interface is used to control dependencies between components. An interface is provided by a component and can be used by other components. Once an interface is changed, it affects all the components that require it directly or indirectly. So interface stability is very important.

In ASML, the default language for specifying an interface is the ddf (data definition file) language. From ddf, specific interfaces for other languages can be generated.

For each interface, there is lots of information. For example, the files it contains, the level it is provided at, the level it is shared at. In this project, such information is called interface property. IMAS Interface Viewer is an ASML internal tool that is aimed to analyze interface properties.
4.4 Software Release

A software release is [6]:

- A set of executables to bring the functionality of the software archive to the customers
  - To run production
  - To set up and recover the machine
- A set of documents:
  - User manuals
  - Change manual
  - Reference manuals

For different TWINSCAN machine types, there are different software releases. Apart from these releases for products, another software archive exists which could be seemed as the mother of all product releases, the QBL release. In the techwiki of ASML, the QBL release is well described:

In order to prevent having to start from scratch with each TWINSCAN release, a Qualified Base Line (QBL) exists. The QBL is a software archive that supports every TWINSCAN machine type. It is the basis for every release. It is some sort of 'software aorta', in the sense that every release is split off from the QBL at some moment in time, while the QBL keeps on 'flowing'. A split off from QBL is what happens when a release becomes version alpha. Until this split off the respective development release is identical to QBL, so deliveries meant for that release are done on QBL. After split off, changes to the software have to be made by patching. [7]

In Figure 6 the process is shown schematically.

![Figure 6 Split Off of QBL](image-url)
4.5 Data Model

In previous sections, the CPD application, interface, and software release are introduced. The relationships between them are:

- One CPD application may exist in multiple releases.
- One CPD application requires a list of external interfaces. The interfaces required by one CPD application may be different in different releases.

One CPD application has multiple properties, as shown in Table 9, Table 10, and Table 11. Based on whether they could be changed in different releases, these properties are separated into two groups:

- Some properties could be different in different releases. These properties are called release-dependent properties. For example, one CPD application can be moved from one package to another package in different releases. So building block and functional cluster are release-dependent properties.
- Some properties are always the same in different releases. These properties are called release-independent properties. For example, CPD type, which is calibration, performance, diagnostics, analysis and adjustment, is never changed. So CPD type is release-independent property.

One interface has multiple properties, as discussed at the end of Section 4.3.

As a conclusion, there are six basic data elements in this domain: CPD application, interface, release, release dependent CPD properties, release independent CPD properties, and interface properties.

In order to organize and process all these data in a standard way, a data model is designed by using UML modeling language, as shown in Figure 7.

![Figure 7 Data Model](image)

Some variables in the model are explained:

- ri_props: release independent properties
- rd_props: release dependent properties
- props: properties

There are four class elements in the model: data_model, cpd, release, and interface. One data_model contains multiple cpd applications. The relationship between cpd, release, and interface is exactly the same as described at the beginning of the section.
5. Feasibility Analysis

In the previous chapters, problem analysis and domain analysis is explained. This chapter performs the feasibility analysis.

5.1 Issues

5.1.1. Format Change of Data Sources

The analysis of the data sources is highly dependent on their formats. If the format is changed, the analysis would be affected.

Issue mitigation strategy: the toolset can be designed to treat the data sources independently, as illustrated in Figure 8. This design can mitigate the issue in two aspects:

- The format change of one data source would only affect the analysis of that data source. The analysis of other data sources would not be affected.
- If the format of one data source is changed, the corresponding adapter of that data source can be updated to match that data source again.

![Figure 8 Illustration of the Design That Treating Data Sources Independently](image)

5.1.2. Separate ADT Applications from CPD Applications

As explained in Section 4.2, ADT applications are treated as different applications from CPD applications. However, in the software archive, ADT applications and CPD applications are mixed together. There is no standard rule to differentiate ADT applications from CPD applications.

Issue mitigation strategy: Some rules are gathered by talking with experts of CPD applications.
5.2  **Risks**

5.2.1. **Compatibility of Environment Configuration**

The incompatibility of user’s environment configuration would affect the use of the toolset. Some incompatibilities would be:

- The version of the application/library is lower than required.
- The Application/library is not supported.

These incompatibilities would cause the toolset to be unable to run. In order to mitigate the risk, two strategies can be used:

- Configure the environment during installation. Check and install all the required applications/libraries when users install the toolset.
- Configure the environment during run-time. The environment may change during run-time. The toolset itself should check the configuration when the environment is changed and install all necessary applications/libraries.
6. System Requirements

The translation from the needs of clients to software requirements is very critical for the success of a software product. Unclear or incomplete requirements lead to the failure of a product.

6.1 Introduction

The requirement gather process is a process of discussing with relevant stakeholders and analyzing the problem and domain. With this process going on, the image of the toolset is more and more clear.

The requirements are divided into three groups: functional requirements, non-functional requirements and domain requirements. For each requirement, a priority is assigned to it.

6.2 Product Perspective

As introduced in Chapter 3, the toolset is aimed to access the data sources of CPD applications and perform analysis to get the required properties. It is developed as a standalone application works in the Linux VDI. Inside the toolset there is an internal database to store the processed data.

The context diagram is shown in Figure 9.

![Figure 9 Context Diagram](image)

6.3 Main Features and Use Cases

Three main features are identified from the requirement gather process:

1. Query management.
   A query is a standard SQL statement that can select data from the internal database. There are multiple queries and each query is identified by its name. The user can manage queries with the following operations:
   - Create a query: create a new query item.
   - Edit a query: edit a query statement.
- Rename a query: change the name of a query
- Copy a query: copy the query statement
- Paste a query: create a new query which is exactly the same as that is copied.
- Delete a query: delete a query
- Save a query: save the query in order to be reused
- Run a query: execute the query statement. The result are saved to a file.
- Check the result of a query.
- Create a user-defined function as a query to combine and output the data with a specific format, such as XML format.

2. Database management.
The user can manage the internal database with the following operations:
- Check the table and column information. The information would be helpful for user to edit the query.
- Flush the database. User can check the data size and flush the database.
- Save the database. Save the data in the database in order to use it at a later time.

3. Converter management.
The processed data in the internal data is converted from the data sources. Converter is the tool that performs the process of converting. The user can manage the converter with the following operations:
- Configure the converter
- Start the converter
- Stop the converter

Based on the main features, the system level use case is shown in Figure 10.
6.4 Functional Requirements

Based on the main features, the functional requirements are listed in Table 12, Table 13, and Table 14, grouped by the main feature.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_1.1</td>
<td>The user can create a new query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.2</td>
<td>The user can edit a query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.3</td>
<td>The user can save a query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.4</td>
<td>The user can delete a query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.5</td>
<td>The user can run a query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.6</td>
<td>The user can rename a query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.7</td>
<td>The user can copy and paste a query in the GUI</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.8</td>
<td>The result of running a query should be saved to a file</td>
<td>High</td>
</tr>
<tr>
<td>FR_1.9</td>
<td>The user can check running result of a query in the GUI</td>
<td>Medium</td>
</tr>
<tr>
<td>FR_1.10</td>
<td>The user can create a user-defined function to combine and output data with a specific format.</td>
<td>Medium</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_2.1</td>
<td>The user can get meta data of internal database</td>
<td>High</td>
</tr>
<tr>
<td>FR_2.2</td>
<td>The user can flush the internal database</td>
<td>High</td>
</tr>
<tr>
<td>FR_2.3</td>
<td>The user can save the data in the internal database.</td>
<td>High</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR_3.1</td>
<td>An configuration file should be provided for the user to configure the converter</td>
<td>High</td>
</tr>
<tr>
<td>FR_3.2</td>
<td>The user can start and stop the converter</td>
<td>High</td>
</tr>
</tbody>
</table>
| FR_3.3 | Two running modes can be provided:  
1. Single running mode. Converter stops after converting the data into internal database.  
2. Continuous running mode. Converter runs in a loop and continues converting. | Medium |

6.5 Non-Functional Requirements

The most important non-functional requirement is extendibility. The toolset should be able to be extended in the following aspects:

- The toolset can be extended to analyze more CPD properties. This gives freedom and convenience to CPD domain experts to analyze CPD properties.
- The toolset can be extended to output the data result with user-defined formats, for readability or for further process.

6.6 Domain Requirements

Domain requirements are the requirements which come from the application domain and reflect the characteristics of that specific domain. In this project, the domain is
CPD application and the domain requirements are mainly relevant to CPD properties. Table 15 shows an initial collection of the domain requirements.

**Table 15 Domain Requirements**

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR_001</td>
<td>For a CPD application, analyze its required interface(s)</td>
<td>High</td>
</tr>
<tr>
<td>DR_002</td>
<td>For an interface, analyze its change times</td>
<td>High</td>
</tr>
<tr>
<td>DR_003</td>
<td>For an interface, analyze to which layer the interface belongs to</td>
<td>High</td>
</tr>
<tr>
<td>DR_004</td>
<td>For a CPD application, identify its functional cluster and building block</td>
<td>High</td>
</tr>
<tr>
<td>DR_005</td>
<td>For a CPD application, identify its type: calibration, performance or diagnostics</td>
<td>High</td>
</tr>
<tr>
<td>DR_006</td>
<td>For a CPD application, analyze the release(s) in which it is created and modified</td>
<td>High</td>
</tr>
<tr>
<td>DR_007</td>
<td>For a release, get a list of all the CPD applications in that release</td>
<td>High</td>
</tr>
<tr>
<td>DR_008</td>
<td>For a CPD application, analyze weather it supports scenarios or not</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_009</td>
<td>For a CPD application, analyze weather it supports sequence runner or not?</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_010</td>
<td>For a CPD application, analyze which framework it uses</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_011</td>
<td>For a CPD application, analyze which libraries it is used</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_012</td>
<td>For a CPD application, analyze it is programming language</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_013</td>
<td>For a CPD application, analyze the number of its UI elements</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_014</td>
<td>For a CPD application, analyze its running duration time</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_015</td>
<td>For a CPD application, analyze its selected scenarios at run-time</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_016</td>
<td>For a CPD application, analyze its running result</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_017</td>
<td>For a CPD application, analyze the number of its steps</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_018</td>
<td>For a CPD application, analyze the modified files and functions</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_019</td>
<td>For a CPD application, analyze the lines of generated code</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_020</td>
<td>For a CPD application, analyze the lines of handcrafted code</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_021</td>
<td>For a CPD application, analyze the size of used data</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_022</td>
<td>Identify patterns in CPD: Prepare, Measure, Calculation, Report, Save Calibration</td>
<td>Middle</td>
</tr>
<tr>
<td>DR_023</td>
<td>For a CPD application, analyze its EIoC.</td>
<td>Low</td>
</tr>
<tr>
<td>DR_024</td>
<td>For a CPD application, analyze its TQI metrics.</td>
<td>Low</td>
</tr>
<tr>
<td>DR_025</td>
<td>For a CPD application with MATLAB code, analyze the method to integrate MATLAB code: embedded or translated</td>
<td>Low</td>
</tr>
<tr>
<td>DR_026</td>
<td>For a CPD application, analyze its calling stack</td>
<td>Low</td>
</tr>
<tr>
<td>DR_027</td>
<td>For a CPD application, analyze its interaction with operating system</td>
<td>Low</td>
</tr>
<tr>
<td>DR_028</td>
<td>For a CPD application, analyze the lines of added code when it is modified</td>
<td>Low</td>
</tr>
<tr>
<td>DR_029</td>
<td>For a CPD application, analyze the lines of deleted code when it is modified</td>
<td>Low</td>
</tr>
<tr>
<td>DR_030</td>
<td>For a CPD application, analyze the lines of changed code when it is modified</td>
<td>Low</td>
</tr>
<tr>
<td>DR_031</td>
<td>For a CPD application, analyze the lines of changed code when it is modified</td>
<td>Low</td>
</tr>
</tbody>
</table>
7. System Architecture

In the previous chapter, the system requirements are described. This chapter describes how the toolset is decomposed into components and how these components interact with each other.

7.1 Introduction

Like architecture for a building, software architecture defines the structure and/or behavior of the software system. Typically architecture decomposes the system into components and defines the relationships and interfaces between these components. Usually the decomposition is an iterative process which is from a general, high-level functional structure to more detailed and concrete descriptions.

In this project, the architecture of the toolset is designed based on the following key drivers:

1. Design constrains
   As described in Section 3.3, the toolset is a standalone application. As a result, no network communication is needed and the toolset itself is a thick client with all the functionalities.

2. Non-functional requirements
   As described in Section 6.5, extendibility is the most important non-functional requirements.

3. As analyzed in section 3.4 an internal database is used in the toolset.

4. As analyzed in section 3.2, a graphical user interface should be provided, through which the user can interact with the toolset.

5. Functional requirements
   As described in Section 6.3 and Section 6.4, there are three main functionality features: query management, database management and converter management.

Based on the third and fourth key drivers described above, a database and a frontend should be included in the toolset. A converter is also needed to transform the raw data into the internal database. The frontend, the database and the converter are the three main elements in the toolset. Figure 11 shows the top level architecture.

![Top Level Architecture](image)

**Figure 11 Top Level Architecture**

The frontend is the graphical user interface. Via the frontend, user can manage queries, manage the internal database and control the converter.
The converter is to access the multiple raw data sources, parse the raw data and store the processed data into internal database. The internal database is the storage of processed data. In the following sections, the internal architectures of the frontend and the converter are described.

### 7.2 Converter

The architecture of converter is shown in Figure 12.

![Figure 12 Architecture of Converter](image)

The converter behaves as a data filter and transformer between the raw data sources and the internal database. Filter and transform are the two main functionalities of the converter, which are explained below:

- **Filter.** Extract the required properties from the raw data sources.
- **Transform.** Transform the required properties into the internal database

Corresponding to the two functionalities, two components are introduced in the architecture: the adapter and the transformer.

The adapter can be property oriented that for each property, there is an adapter. With this design, extendibility of the toolset can be achieved. In order to analyze a new property, a new adapter can be added.

The data model, which is described in Section 4.5, behaves as the interface between the transformer and the adapters. Each adapter should encapsulate the property into the data model and return it to the transformer. The benefit of this design is that it decouples the transformer from the adapters:

- For the multiple adapters, only one transformer is needed to transform the property into internal database. The transformer is only interested in the data model. It does not care how the adapter is implemented and how many adapters are there.
- The transformer is not affected by adding a new adapter, because each adapter returns the same data structure – the data model to the transformer.

Figure 13 shows the data flow in the converter. First the raw data is extracted into the data model by the adapter. Then the data model is transformed into the internal database by the transformer.
A configuration file is also provided in the architecture. The configuration file is aimed to provide some options for the user to select. The available options are:

- **releases**: a list of view names in which the user would like to perform the analysis
- **cpds**: a list of CPD application names of which the user would like to get the property
- **cpd properties**: a list of CPD properties that the user would like to get
- **interface properties**: a list of interface properties that the user would like to get

These options are used to select the corresponding adapters or used as parameters when performing the analysis.

### 7.3 Frontend

Figure 14 shows the architecture of the frontend.

**Figure 14 Architecture of Frontend**

Corresponding to the three main functionality features, as described in Section 6.3, there are three components in the frontend:

- **Converter management**: This component provides the user interface to start, stop and configure the converter. User can also get the status of the converter via this component.
- **DB management**: This component provides the user interface to manage the internal database and get the metadata.
- **Query management**
This component provides the user interface to manage the queries and get data from internal database. The user can maintain a list of pre-defined queries.

In the frontend there are two data storages:

- **Query result.**
  The running result of each query is saved into a file. The user can check the running result of a query by opening the result file.

- **Query storage.**
  Each query is a SQL statement. These statements are saved into a file in order to be reused.
8. System Design

In the previous chapter, the high level system architecture is introduced. The frontend, the converter and the internal database are the three main elements. This chapter gives more details about the design of the three main elements.

8.1 Introduction

In this chapter, UML\(^1\) modeling language is used to describe the design of the toolset and Enterprise Architect\(^2\) is used as the tool for UML models. Three kinds of diagrams are used:

- Class diagram\(^3\): used to describe the structure of the system
- Sequence diagram\(^4\): used to describe the behavior of the system
- Data model diagram: extend class diagram used to described the structure of a relational model

In Enterprise Architect, there is a functionality call Code Engineering that is able to:

- Generate code from UML model
- Generate UML model from code

It is used in the design and implementation of the toolset.

For the converter, a class diagram is designed to describe the classes, attributes, methods and the relationships between the classes. Code Engineering is used to generate the structure code from the class diagram. In this way, the implementation and the UML model are consistent with each other.

For the internal database, a data model diagram is designed to describe the tables and columns. Code Engineering is used to generate the SQL statement of database creation.

The frontend is implemented based on a mockup, which is shown in Figure 18. Code Engineering is used to generate the class diagram from the code to show the main classes and the relationships between the classes.

8.2 Converter

Figure 15 shows the class diagram of converter.

The class diagram of the converter is designed based on elaborating the components in the architecture of the converter. Corresponding to the two main components shown in the architecture (Figure 12), there are three main classes in the class diagram.

Transformer class is designed based on Transformer component. It has two main methods:

- convert: convert data model into internal database
- get_config: get user configuration

It refers to RdAdapterAdmin class to access the adapters.

---

RdAdapterAdmin class and RdAdapter class are designed based on Adapter component. RdAdapterAdmin class is the administrator of all the RdAdapter classes. Every RdAdapter class is registered via register_adapter method and can be obtained via get_adapter method. RdAdapter class is a virtual base class that all the concrete adapter classes are derived from it.

There are two design patterns [8] used in this design:

- **Singleton pattern**
  
  RdAdapterAdmin is the entrance to all adapters. In the toolset, there should be only one entrance of adapters and the entrance should be easily accessible. So RdAdapterAdmin is designed as a singleton class in order to ensure that:
  
  1. Only one instance of RdAdapterAdmin exists in the toolset
  2. The instance of RdAdapterAdmin can be accessed easily.

- **Strategy pattern**

  For each adapter, there is an algorithm of parsing the raw data to get the required properties. In this toolset, transformer is the client that would use these algorithms. Strategy pattern is used to decouple these algorithms from the client. Each algorithm is encapsulated into the method get_data_model. It brings the following benefits:
  
  - The algorithms vary independently from the client.
  - Only selected algorithms would be executed in the run-time.
  - It is easy to add or delete algorithms with no effect on the client.

**Figure 16** shows the sequence diagram of converter. It depicts the behavior of converter. The normal work flow is described in the following.

First, the transformer gets the user configuration. Then it gets the instance of RdAdapterAdmin. Next is a loop of instantiate and execute adapters based on user configuration to get the required properties. When the execution of all the adapters is completed, the transformer converts the data into internal database.
8.3 Internal Database

As shown in Figure 13, the data in the internal database is transformed from the data model. The internal database and the data model contains the same data, but with different format. The data model shows the relationships between the data elements. The database is designed based on the data elements and their relationships.

Figure 17 shows the design model of internal database.

There are four tables in the database, which correspond to the four data elements: release dependent CPD properties, release independent CPD properties, interface and interface properties.

These tables can be extended by adding a new column when a new property is to be analyzed:

- If it is release dependent CPD property, a new column can be added to the table cpd_rd_property.
- If it is release independent CPD property, a new column can be added to the table cpd_ri_property.
- If it is interface property, a new column can be added to the table interface_.

![Sequence Diagram of Converter](image-url)
8.4 Frontend

Figure 18 shows the mockup of the user interface. The converter management and the database management are in the top menu bar. Query management is a treeview structure with context menu to run the commands. The status of the converter is shown in the top right of the window.
Figure 19 shows the class diagram of the frontend.

The classes in the class diagram fulfill the functionality of the components in the architecture of the frontend (Figure 14):

- **Query management.**
  The query management component is designed as a tree view structure, which is the QueryTree class in Figure 19. Compared to a list structure, a tree view structure gives the user more freedom of organizing the queries. All the query operations are performed via the context menu.

- **Converter management.**
  The converter management component is designed as a top menu item. With the submenu items, the user can start, stop and configure the converter. When the converter starts, it runs in a new process. So the converter and the frontend are able to run in parallel. In order to show the running state of the converter, a state machine is designed, as shown in Figure 20.

There are four states in the state machine.

As shown in Figure 13, ‘adapting’ and ‘transforming’ are the two main working steps in the converter. In these two steps, the converter has different impacts on the frontend:

- In the ‘adapting’ step, the converter is parsing the raw data to get the required properties and has no impact on the frontend.
- In the ‘transforming’ step, the converter is writing data into the internal database. The internal database is the intersection between the converter and the frontend. Because the database is updating, the frontend is blocked from accessing the internal database in this step.

So in the state machine, there are two states corresponding to the two steps: the ‘adapting’ state and the ‘transforming’ state. The ‘stopped’ state in the state machine is to denote the state when the converter is not working.
The ‘prepare transforming’ state behaves as a time buffer between the ‘adapting’ state and the ‘transforming’ state. It is used to notify the user that the ‘transforming’ state is coming and the frontend is blocked from accessing the internal database.

Table 16 shows the state transition table. The user can stop the converter in the ‘adapting’ state and ‘prepare transforming’ state, but not in the ‘transforming’ state. The reason is that, in the ‘transforming’ state, the internal database is updating with new data and stop the converter would cause incomplete data in the database.

- Database management.
  The database management component is designed as a top menu item. As shown in Figure 21, a popup window is used to show the metadata.
The behavior model of the three main functionality features is explained in the following.

Figure 22 shows the sequence diagram of query management in the frontend. First the user right-clicks on a node item in the query tree to show the context menu. Then the user left-clicks on a menu item to execute that command.

Figure 22 Sequence Diagram of Query Management in the Frontend

Figure 23 shows the sequence diagram of the converter management in the frontend. First the user left-clicks on the top menu item of converter to show the sub menu items. There are three sub menu items: configure converter, start converter and stop converter. Each sub menu item is bound to a function that is implemented in the class ConverterAction.

Figure 23 Sequence Diagram of Converter Management in the Frontend
Figure 24 shows the sequence diagram of the database management in the frontend. First the user left-clicks on the top menu item of database management to show the sub menu items. There are two sub menu items: show metadata, flush database. Each sub menu item is bound to a function that is implemented in the class DBManagement.

![Sequence Diagram of Database Management in the Frontend](image-url)
9. Implementation

In the previous two chapters, the system architecture and system design are discussed. This chapter describes the implementation details.

9.1 Introduction

The implementation is based on the system architecture and system design, as described in the previous two chapters, and satisfies the system requirements. The implementation is provided as a prototype to demonstrate the feasibility of the project.

Some technologies are chosen to be used in the implementation.

Python\(^5\) is used as the main programming language in the development of the toolset, mainly based on the following reasons:

- It is cross-platform
- It is easy to learn and use
- It is easy to perform data processing

TKInter\(^6\) is used as the GUI framework to develop the user interface. It is Python's de-facto standard GUI and the mostly commonly used GUI programming toolkit for Python.

SQLite\(^7\) is used as the database engine. It is not a client-server database engine, but embedded into the end program. It is highly useful for standalone applications.

9.2 Converter

The design of the adapter in the converter is to make the toolset extendible. A new adapter can be implemented when a new property is to be analyzed.

Five adapters are implemented, as shown in Table 17. The column names in Table 17 are explained:

- Adapter: the adapter name.
- Property: the property that is obtained by the adapter.
- Data source: the data source from where the adapter gets the property.
- Requirement: the requirement that is fulfilled

<table>
<thead>
<tr>
<th>Adapter</th>
<th>Property</th>
<th>Data source</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPTM Adapter</td>
<td>CPD name, CPD type</td>
<td>XPTM files</td>
<td>DR_006, DR_008</td>
</tr>
<tr>
<td>Programming Language</td>
<td>CPD programming language</td>
<td>Source files</td>
<td>DR_013</td>
</tr>
<tr>
<td>Language Adapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface Adapter</td>
<td>CPD required interface</td>
<td>Scope files / source files</td>
<td>DR_001, DR_011</td>
</tr>
<tr>
<td>Package Adapter</td>
<td>Package/building block, functional cluster</td>
<td>PKG_contents file</td>
<td>DR_005</td>
</tr>
<tr>
<td>Stream Adapter</td>
<td>History streams where a CPD application has been created and modified</td>
<td>Source files</td>
<td>DR_007</td>
</tr>
</tbody>
</table>

---

\(^5\) https://www.python.org/

\(^6\) https://wiki.python.org/moin/TkInter

\(^7\) https://www.sqlite.org/
All the five adapters except the XPTM adapter can be used to get the property not only for CPD applications, but for all the applications in the TWINSCAN software archive. For example, the programming language adapter can be used to get the programming language property of any application in the software archive. So the scope of using the toolset is not strictly limited to only CPD applications. XPTM adapter can only be used for CPD applications, because XPTM file is a specific file for test applications, which include CPD applications and ADT applications.

Several domain requirements are fulfilled by the implementation of the adapters, as shown in Table 17. The fulfillment of these requirements demonstrates the feasibility of the toolset. The implementation of the five adapters also demonstrates the extendibility of the toolset. It is described in Chapter 11 that how to extend the toolset.

### 9.3 Internal Database

In Section 8.3 the database model is described. As described in Section 8.1, the SQL statement of database creation is generated automatically from the database model by using Enterprise Architect. Figure 25 shows the SQL statement of database creation.

```sql
DROP TABLE IF EXISTS interface;
CREATE TABLE interface
(
    interface_id INTEGER NOT NULL PRIMARY KEY,
    interface_name TEXT NOT NULL,
    release TEXT NOT NULL,
    CONSTRAINT uniq UNIQUE (interface_name, release)
) ;

DROP TABLE IF EXISTS cpd_interface;
CREATE TABLE cpd_interface
(
    cpd_interface_id INTEGER NOT NULL PRIMARY KEY,
    cpd TEXT NOT NULL,
    release TEXT NOT NULL,
    interface_name TEXT NOT NULL,
    CONSTRAINT uniq UNIQUE (cpd, release, interface_name)
) ;

DROP TABLE IF EXISTS cpd_rd_property;
CREATE TABLE cpd_rd_property
(
    cpd_rdprop_id INTEGER NOT NULL PRIMARY KEY,
    cpd TEXT NOT NULL,
    release TEXT NOT NULL,
    id TEXT,
    package TEXT,
    CONSTRAINT uniq UNIQUE (cpd, release)
) ;

DROP TABLE IF EXISTS cpd_rl_property;
CREATE TABLE cpd_rl_property
(
    cpd_rlprop_id INTEGER NOT NULL PRIMARY KEY,
    cpd TEXT NOT NULL,
    top TEXT,
    title TEXT
) ;
```

**Figure 25 SQL Statement of Database Creation**

### 9.4 Frontend

Figure 26 shows the implementation of the user interface. It is implemented according to the mockup, as shown in Figure 18, and fulfills the functional requirements in Section 6.4.

The converter management, database management and help content are in the top menu bar. The query management is implemented in the treeview structure on the left
side of the user interface. In the treeview structure, the queries can be organized into different folders. The running status of the converter is shown on the top right side.

![Figure 26 User Interface](image)

In the query manager, the user can create query to get data in the database. There are two kinds of queries:

- **Single SQL statement.**
  It selects specific columns from the internal database and the output data is in the format of columns separated by tabs.

- **Self-defined function.**
  SQLite does not support stored procedure, which executes a batch of SQL statements. However, with SQLite, user can create self-defined functions in the host language, which is Python in this project, to execute a batch of SQL statements and combine the data together in a specific format.

Figure 27 shows one output example of self-defined function. It calculates the number of CPD applications in each package by language.

For how to create and use self-defined function, please refer to Section 11.3.3.

![Figure 27 Output Example of Self-Defined Functions](image)
10. Verification & Validation

In the previous chapters, the design and implementation of the toolset are described. This chapter describes the verification and validation of the toolset.

10.1 Verification

Verification is the evaluation of whether or not a product complies with the requirement and specification.

In this project, there are three kinds of requirements: functional requirements, non-functional requirements and domain requirements. Verification is performed for each kind of requirements.

The functional requirements are verified manually. Each operation is executed and the result is checked. Some bugs are found and fixed. Due to time constraints, no formal-written test cases are generated and unit test is not performed.

Extendibility is the most important non-functional requirement in this project. The toolset is a framework with adapters as plugins. It is verified in the following way:
- Five adapters, which are targeted for different CPD properties, are implemented. It demonstrates that the toolset is extendible that adapters can be added to the toolset to extend it.
- One adapter is implemented by the client and works correctly. It demonstrates that it is not difficult to extend the toolset.

The domain requirements are verified by manually checking the data result.

10.2 Validation

Validation is the evaluation of whether or not a product meets the needs of the stakeholders.

Two ways are used in this project to validate the toolset.

The first way is to apply the toolset to analyze the interface stability, which is the most important and interesting topic for the clients. At the time when this report is completed, the work of interface stability analysis is not completed (see the Gantt chart in Figure 38). The planned tasks of the interface stability analysis are described in Section 12.1.

The second way is to collect feedback from the stakeholder. An evaluation form is created and the result is shown in Table 18.

<table>
<thead>
<tr>
<th>Questions</th>
<th>No opinion</th>
<th>1 (poor)</th>
<th>2 (fair)</th>
<th>3 (good)</th>
<th>4 (very good)</th>
<th>5 (excellent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Experience</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How easy is it to manage the queries in the frontend</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>How easy is it to use the converter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Extendibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How easy is it to add a new adapter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>How easy is it to add a user-defined SQLite function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>General</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How helpful is the toolset in your</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
work

| How easy is it to learn to use the toolset | x |
| How did you like the overall design of the toolset | x |
| How did you like the overall performance of the toolset | x |

**General comments**

The converter editor could be made more user friendly when selectable elements (drop down lists) would have been created i.s.o. a plain editor.
This is also the case for the query editor although this would become a bit more complex.
Another small improvement could be made when the results were shown in the frontend directly after a query was run.
What I’m missing is an auto tester which can be run to check the requirements.
But, this tool will fit its purpose well, and it is extendable for new features. I’m looking forward in using this tool. It will help us a lot!

The feedback could be divided into tips and tops:
- **Tips** – The user interface could be improved to be more user friendly. An auto tester is needed to check the requirements.
- **Tops** – The toolset will fit its purpose well and be very helpful in his work. The functionality and extendibility are good.

As a conclusion, the toolset meets the needs of the stakeholder and there are rooms for improvement.
11. User Guide

The design and implementation of the toolset are described in the previous chapters. This chapter introduces how to use the toolset and how to extend it.

11.1 Introduction

CPD Analysis Toolset is developed to perform analysis on CPD applications and get the properties. It is developed using Python. It is a standalone application running in the Linux VDI environment in ASML.

Following CPD properties are available in the latest version of the toolset:

- Package. The functional cluster, building block and package information of a CPD application.
- Language. The programming language used by a CPD application
- Type. The CPD application type, which is calibration, performance, diagnostics, analysis and adjustment.
- Required interface. External interface(s) required by a CPD application
- History stream. The streams in which a CPD application has been created and modified.

11.2 How to Use the Toolset

11.2.1. Installation

$ cp /home/dohu/cats_release/cats_install .
$ ./cats_install

11.2.2. Start the Toolset

$ cd ./cats
$ ./run &

11.2.3. Query Management

In the query manager, user can manage queries in the tree structure. There are two types of items with different icons:

- Folder: item with .

  Each folder can contain multiple folders and queries. By right clicking on a folder item, a context menu shows up as below:

  ![Context Menu of Folder Item]

  **Figure 28 Context Menu of Folder Item**

  The following operations are available via the folder context menu:
- Create new query
- Create new folder
- Rename the folder
- Paste query
- Delete folder. When a folder is deleted, all its children (the folders and queries it contains) are deleted.

- Query: item with icon.
  Each query item contains a query statement. By right clicking on the query item, a context menu shows up as below:

  ![Context Menu of Query Item](image)

  **Figure 29 Context Menu of Query Item**

  All the following operations are available via the query context menu:

  - Run the query
  - Rename the query
  - Delete the query
  - Save the query
  - Edit the query
  - Copy the query
  - Check the result

  Double click on a query item can also open the ‘edit query’ dialog box.

  The data result of each query is saved into a file, which is named as `<query name>_<dd>-<mm>-<yyyy>`. The query result files are in the folder “query_res”.

  A query is a standard SQL statement. One query example is shown below:

  ```sql
  select interface_name from cpd_interface where cpd = 'XYZ' and release_ = 'qbl'
  ```

  This query gets all the required interfaces of ‘XYZ’ in the latest QBL. Here ‘XYZ’ is the name of a CPD application, and ‘qbl’ is the name of a release, which means the latest QBL. If the user would like to query data from a specific QBL release, he use the full name.

  For more knowledge of SQLite query, please refer to:

11.2.4. Converter Management

Converter is the tool that can parse the data source and store the processed data into the internal database. The user can start, stop and configure converter via Frontend. A closer view of the converter menus are shown in Figure 30.
Configure Converter

Figure 31 shows the configuration items of converter, which are explained below:

- **releases**: in which releases the user would like to get the CPD data. Each release item is a ClearCase view name, or ‘qbl’, which means the latest QBL view.
- **cpds**: a list of CPD names. The option ‘All’ means all CPDs that are available.
- **cpd_properties**: the CPD properties that the user would like to get
- **interface_properties**: the interface properties the user would like to get

```
# Each config item is a key value pair.
# Keys and values are not case sensitive.
# Space around the ':' is allowed.
# A value can be a string list in which the strings are separated by commas.
# Values can be in multiple lines.
# Each line begin with the equal-colon(':') is the end of a config item.
# Each line begin with the hash character ('#') is a comment.

releases;
#
cpd names:
cpds: all
;
# FC, EB, c_p_d_type, created_release, changed_releases
 cpd_properties:
;
# name, change_times, layer, level
 interface_properties:
```

Start Converter

User can start converter to parse the data by pressing the submenu 'start', which is under the menu 'Converter', as shown in Figure 30.

When the converter is started, the converter status is shown as below:

```
Converter Status

Converting
```
Stop Converter

User can stop converter by pressing the submenu ‘stop’, which is under the menu 'Converter', as shown in Figure 30. When the converter is stopped, the converter status is shown as below:

![Converter Status](image)

**Figure 33 Stopped Status of Converter**

11.2.5. Database Management

The user can perform database operations via the menus of database management. The menus are shown below:

![Menus of Database Management](image)

**Figure 34 Menus of Database Management**

Via the ‘Flush DB’ submenu, the user can flush the internal database and check the size of it. Figure 35 shows the message box which pops up when the user presses this submenu.

![Message Box of 'Flush DB' Submenu](image)

**Figure 35 Message Box of ‘Flush DB’ Submenu**

Via the ‘Show Info’ submenu, the user can check the table and column information. Figure 36 shows the dialog box which pops up when the user presses this submenu.

![Dialog Box of 'Show Info' Submenu](image)

**Figure 36 Dialog Box of ‘Show Info’ Submenu**
11.3 How to Extend the Toolset

11.3.1. Add a New Adapter of CPD Property

A new CPD property can be analyzed by adding a new adapter. Here explains how to add a new adapter of CPD property.

1. A name should be given to the property. Here it is named as ‘xyz’
2. Add a new file in the folder ‘src/Converter/Adapter/’, named as ‘xyzAdapter.py’
3. In the file ‘xyzAdapter.py’, create a derived class ‘xyzAdapter’ from ‘RdAdapter’ and implement the method ‘get_data_model’. Below is an example

   ```python
class xyzAdapter(RdAdapter):
   def get_data_model(self, releases, cpds, model):
       self._model = model
       # add your code here
       return self._model
   ```

4. In the file ‘src/Converter/RdAdapterAdmin.py’, add the following line at the end of the method ‘register_adapter’:

   ```python
   self._adapters["xyz"] = xyzAdapter()
   ```

5. In the file ‘cfg/converter.cfg’, add ‘xyz’ to the comment line above the CPD properties configuration line. So the user knows this CPD property is available.

6. If ‘xyz’ is release dependent property, add a new column named ‘xyz’ to the table ‘cpd_rd_property’. If ‘xyz’ is release independent property, add a new column named ‘xyz’ to the table ‘cpd_ri_property’. For the information of database, please refer to Section 8.3. For how to add a new column, please refer to Section 11.4.

11.3.2. Add a New Adapter of Interface Property

A new interface property can be analyzed by adding a new adapter. To add a new adapter for a new interface property, the steps are:

1. A name should be given to the property. Here it is named as ‘xyz’
2. Add a new file in the folder ‘src/Converter/Adapter/’, named as ‘xyzAdapter.py’
3. In the file ‘xyzAdapter.py’, create a derived class ‘xyzAdapter’ from ‘RdAdapter’ and implement the method ‘get_data_model’. Below is an example:

   ```python
class xyzAdapter(RdAdapter):
   def get_data_model(self, releases, cpds, model):
       self._model = model
       # add your code here
       return self._model
   ```

4. In the file ‘src/Converter/RdAdapterAdmin.py’, add the following line at the end of the method ‘register_adapter’:

   ```python
   self._adapters["xyz"] = xyzAdapter()
   ```
5. In the file ‘cfg/converter.cfg’, add ‘xyz’ to the comment line above the interface properties configuration line. So the user knows this interface property is available.

6. Add a new column named ‘xyz’ to the table ‘interface_’. For the information of database, please refer to Section 8.3. For how to add a new column, please refer to Section 11.4.

11.3.3. Create Self-Defined SQLite Function

About how to create user defined SQLite functions in Python, please refer to the method ‘create_function’ in the following link:
https://docs.python.org/2/library/sqlite3.html

In the toolset, the self-defined SQLite functions are defined in the file ‘src/Common/DBFunctions.py’. 
Steps to add a self-defined SQLite function:
- In the file ‘src/Common/DBFunctions.py’, define your SQLite function. For example, add the following lines to define a new function:

```
def report_xyz(param1):
    out_str = ""
    # add your code here
    out_str = "hello, report_xyz"

    return out_str
```

- In the file ‘src/Common/Database.py’, add the following line at the end of the method ‘register_functions’:

```
self.conn.create_function("report_xyz", 1, report_xyz)
```

- The three parameters of the method ‘create_function’ are explained:
  - First parameter is the name which is used in the SQLite statement. For example, here ‘report_xyz’ is registered. It can be used in SQLite statement like: ‘select report_xyz(<param>)’. <param> is the parameter which should be passed to the function ‘report_xyz’.
  - Second parameter is the number of parameters.
  - Third parameter is the name of the function define in Python

- In the query manager, create a new query item with the following content:

```
select report_xyz(""
```

After running the query, the data result would be the following:
```
hello, report_xyz
```

The query result is exactly the same as the value of the string variable returned by the self-define function.
11.4  SQLite Database Operation

11.4.1. Add New Column

The shell script ‘add_column’ in the folder ‘data/’ can be used to add a new column in the database.

Syntax:
./add_column  <table name>  <column name>  <column type>

There are three parameters for the script:
- table name: the name of the table in which you would like to add the column.
- column name: the name of the column you would like to add.
- column type: the data type of the column you would like to add.

Example:
./add_column  cpd_ri_property  property_1  TEXT
12. Conclusions

In this chapter, the results achieved in this project and future work are elaborated. The lessons learned in this project are also described.

12.1 Results & Future Work

The main result in this project is the toolset which can be used to get the properties of CPD applications. It provides an automatic way to gather data of CPD applications. The toolset is well designed that it is easy to be extended. Several adapters are implemented to demonstrate the extendibility of the toolset. The toolset is already used by the stakeholder in ASML and an adapter is successfully implemented by the stakeholder to get a new CPD property. The feedback of the stakeholder is described in Section 10.2.

Showing a demo to the stakeholders is usually a useful way to convince them of the capability of the product and the feasibility of the approach. In this project, several demos have been shown to the stakeholders. Two demos were shown to the supervisors in the project steering meetings: the first demo demonstrated the configurations and operations of the converter; the second demo demonstrated the operations of the frontend. These demos demonstrated the feasibility of the approach.

One demo was shown to the CPD competence team to demonstrate the usage and capability of the toolset. They showed interests in the toolset and began to use it in their work.

As described in Section 10.2, the tasks of interface stability analysis are planned but are not finished when the report is completed. These planned tasks are:

- Add adapters to the toolset to get the change times of each interface in the last year and the build dependencies of each interface.
- Show the change times of interfaces in a chart.
- For each CPD application, calculate the summary of the change times of its required interfaces.

Several future works could be done on the toolset:

- The toolset is extendible with adding new adapters to get new CPD properties. More adapters could be implemented and added to the toolset. Section 11.3 describes how to extend the toolset.
- Based on the data from the toolset, several analyses could be performed. These analyses could provide insight of CPD applications to the user. There are three ways to perform analysis on the data in the toolset:
  1) Use SQL query to analyze the data. With SQL statements, some calculations could be performed. However, it has its limit for complex analysis.
  2) Create Self-Defined SQLite functions and use Python programming language to analyze the data. Python is a programming language that is good at data processing. There are several libraries could be used, such as SciPy\(^8\), NumPy\(^9\), Pandas\(^10\) and Matplotlib\(^11\). About how to create Self-Defined SQLite functions, please refer to Section 11.3.3.

---

\(^8\) [https://www.scipy.org/](https://www.scipy.org/)
\(^10\) [http://pandas.pydata.org/](http://pandas.pydata.org/)
\(^11\) [http://matplotlib.org/](http://matplotlib.org/)
3) Use third-party tools, such as Excel and MATLAB, to analyze the data.

- The user interface could be improved to be more user friendly. In the ‘general comments’ section in Table 18, some suggestions are provided by the stakeholder. They are:
  1) The converter editor could be implemented as a drop down list which could enable the user to select items.
  2) In the query manager, a drop down list could also be used for the user to select tables and columns.
  3) The result could be shown in the frontend directly after a query was run.

12.2 Lessons Learned

In this project, several lessons are learned. They are summarized in the following.

The first lesson is cooperating with domain experts to solve problems in that domain. It is very helpful to have meetings regularly with domain experts to understand the problem and domain knowledge and receive feedbacks of the solution. It can make sure that the project is in the right direction and really solves the problems.

The second lesson is stepping out of the comfort zone. In this project, it means I should not keep myself in the software domain, but go further in the domain of CPD applications and interface stability. In order to deliver a good project, I have to understand my clients and use their language to communicate with them.

The third lesson is that it is not an easy work to get the domain knowledge, especially when the knowledge is not documented. It makes the communication with domain experts very difficult when different terms are used. Below are some challenges I faced during the process of getting domain knowledge in this project:

- For CPD applications, different names are used. The abbreviations CPD, C&P and CnP have the same meaning. ‘Test software’ is also another name for CPD applications.
- About whether ADT is CPD or not, the CPD experts have different opinions.
- Some domain terms are used by the CPD experts. It takes time to understand the exact meaning of these terms.

The forth lesson is that it is very helpful for the stakeholders that the toolset is not only developed, but also applied to perform analysis in the CPD domain. It is a pity that I have only a very short time to use the toolset to analyze interface stability at the end of the project.

The fifth lesson is using UML model to design a system. UML model shows the structure and behavior of the system in a graphical way that it is very easy to show the design and discuss it with others. It also brings convenience to maintain the application. Code engineering is functionality in Enterprise Architect that is able to generate code from model and generate model from code. It is used in this project and some of the codes are generated from the model.

■
13. Project Management

In this chapter, the project management is described.

13.1 Introduction

At the beginning of the project, an initial project plan is created with the top level tasks. With the project going, the project plan is adjusted and more details are added.

The project steering group meeting, which is held per four weeks, plays a big role in managing the progress of the project. The group consists of the trainee, the university supervisor, the company supervisor and the project manager.

13.2 Work-Breakdown Structure (WBS)

Figure 37 Work-Breakdown Structure of This Project

Figure 37 shows the final revised work-breakdown structure of this project, which decomposes the project into smaller components. Seven top level tasks are identified. Each top level task is decomposed into smaller tasks. These tasks are referenced later on in the project planning section.

The domain study task contains the following tasks:
- Context study – study the context of this project
- Software introduction course – a series of training courses for the software development inside ASML.
- Data source study – study the data sources. Table 4 shows a list of all the possible data sources.
- CPD study – use a CPD application as an example to study the code structure and work flow of CPD applications.

The requirement analysis task contains the following tasks:
- Organize meetings with stakeholders to collect requirements
- Create and revise the CPD property table.
• Create and revise the requirement table. Prioritize the requirements. In Chapter 6 all the requirements are described.

The converter task contains the following tasks:
• Design the converter architecture, which is described in Section 7.2.
• Design the converter model, which is described in Section 8.2.
• Implement the framework.
• Implement the basic adapters, which are described in Section 9.2.

The frontend task contains the following tasks:
• Design the frontend architecture, which is described in Section 7.3.
• Design the frontend model, which is described in Section 8.4.
• Implement the frontend, which is described in Section 9.4.

The database task contains the following tasks:
• Design the database model, which is described in Section 8.3.
• Implement the database, which is described in Section 9.3.

The documentation task is writing the report.

The interface stability task contains the following tasks:
• Adapters implementation – implement adapters to get interface data.
• Data analysis – perform analysis based on the interface data obtained from the toolset.

13.3  Project Planning and Scheduling

13.3.1. Gantt Charts

Figure 38 shows the Gantt chart of this project. It shows that the project contains three periods:
• The first period is domain study and requirement analysis, which are performed in the first three months. In the period, the domain knowledge is studied and an initial list of requirements is collected.
• The second period is system design and implementation, which are performed in the second three months. In this period, the system is divided into three components and each component is designed and implemented.
• The third period is the interface stability analysis, which is performed at the end of this project. In this period, the toolset is applied to analyze the stability status of the interfaces.
13.3.2. Milestone Trend Analysis

At the beginning of the system design phase, some milestones are created, which are shown in Table 19. The first three milestones are relevant to the design and implementation of the toolset. The fourth milestone is relevant to the test of the toolset. The last milestone is finishing the report, which is one important deliverable in this project.

The first three milestones are completed on time.
The fourth milestone is postponed, because:
- Due to time constraints, compared with a fully covered test, documenting and releasing the toolset to the client has a higher priority.
- The deadline of finishing the report is earlier than planned. The task of writing report is brought forward.

The fifth milestone is completed earlier because the deadline is earlier than planned.

<table>
<thead>
<tr>
<th>Milestone ID</th>
<th>Milestone Description</th>
<th>Forecasted Completion Date</th>
<th>Actual Completion Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_01</td>
<td>Complete Architecture of converter;</td>
<td>May 9</td>
<td>May 9</td>
</tr>
<tr>
<td>M_02</td>
<td>Complete Architecture of frontend;</td>
<td>June 6</td>
<td>June 6</td>
</tr>
<tr>
<td>M_03</td>
<td>Complete the implementation of converter;</td>
<td>July 4</td>
<td>July 4</td>
</tr>
<tr>
<td>M_04</td>
<td>Finish Unit test;</td>
<td>August 1</td>
<td>-</td>
</tr>
<tr>
<td>M_05</td>
<td>Finish Report;</td>
<td>August 29</td>
<td>August 19</td>
</tr>
</tbody>
</table>
14. Project Retrospective

In this chapter, a reflection of the project is described from the trainee’s perspective. The three design opportunities that are set at the beginning of the project are revisited.

14.1 Reflection

The project conducted in the past nine months is an experience with a mixture of familiarity, challenge and gain.

The familiar part in this project is the technology. In a previous in-house project, I had experience using Python and MySQL to process and store data. I also studied UML model, which is used to model the toolset in this project, in TU/e.

There are several challenges in this project.

The first challenge is studying the domain of CPD application and software archive in ASML. In this process I got lots of help from many people, including company supervisor, CPD competence team and some colleagues.

The second challenge is following the process. It is my first time to execute a project in such a strictly structural way. Thanks to the help of the supervisors, I managed to follow the process. The benefit of following the process is that the product is able to be traced back from implementation to design and from design to requirements.

Some of the learned lessons are described in Section 12.2. Project management, stakeholder management and communication are my weak points. I am learning and improving these during the project. Domain driven design is another useful concept I have learned in this project.

14.2 Design Opportunities Revisited

Three design criteria that would be applied in this project are described in Section 3.5.

Extendibility is the most important non-functional requirements in this project. The implementation of several adapters by the trainee and the stakeholder shows the extendibility of the toolset.

Self-Descriptiveness evaluates the degree of understandability of source code and design. In this project, the design is clearly illustrated in the system architecture and system model by using enterprise architect. The source code is consistent with the design.

Separation is the last design criteria and is realized by the decomposition of the toolset. The toolset is decomposed into two main components – the frontend and the converter. Each component has its distinct functionality and is decomposed into sub-components. The design of adapter, which puts the functionality of data processing into adapter and separates the toolset from data source, also demonstrates the separation.
# Glossary

## Table 20 Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>Functional Cluster.</td>
</tr>
<tr>
<td>BB</td>
<td>Building Block.</td>
</tr>
<tr>
<td>RP</td>
<td>Release Part.</td>
</tr>
<tr>
<td>AS</td>
<td>Assembly.</td>
</tr>
<tr>
<td>LA</td>
<td>Layer.</td>
</tr>
<tr>
<td>CC</td>
<td>Component.</td>
</tr>
<tr>
<td>CPD</td>
<td>Test Software used during Calibration, Performance and Diagnostics.</td>
</tr>
</tbody>
</table>
Bibliography

References


About the Authors

**Dongqi Hu** received his bachelor degree in Electronic & Information Engineering from Zhejiang University, China, in 2008. His final project is "Design of LCD Controller". He received his master degree from Zhejiang University in 2011. His research field is Embedded System and Multimedia Application, and the diploma thesis is “Testbench of H.264 Hardware Decoder”. He worked in PPS Company, Shanghai in 2011, mainly doing research and developments in video codec for one P2P streaming video network software. He worked in Tuya Company, Beijing, in 2012, mainly doing logic developments for a 3D online FPS game “Army Rage”. From 2013 to 2014, he worked in Galaxy Company, Baoding, mainly focusing on digital picture processing for video surveillance and related applications. His main interests include digital picture processing, multimedia application and embedded software.
4TU.School for Technological Design, Stan Akkermans Institute offers two-year postgraduate technological designer programmes. This institute is a joint initiative of the four technological universities of the Netherlands: Delft University of Technology, Eindhoven University of Technology, University of Twente and Wageningen University. For more information please visit: www.4tu.nl/sai