Setting up a data gathering infrastructure for PRODUKTIV+
Afstudeerverslag

ter verkrijging van de graad van ingenieur
aan de opleiding Informatica
van de Universiteit Twente
op gezag van de voorzitter van de leerstoel Databases
prof. dr. P.M.G. Apers

door

M.D.Schilling

Afstudeercommissie

Dr. ir. M. van Keulen, University of Twente
Dr. R. Müller, University of Twente
F. Badstübner, Infineon Technologies AG
Dr. J. Alt, Infineon Technologies AG
Abstract

In the area of nanoelectronic systems design, the term ‘productivity’ is often used as a scoring metric for project success, but up to now, this variable cannot be measured satisfactorily. Consequently, a reasonably accurate project and capacity planning does not exist in semiconductor design.

The project PRODUKTIV+ deals with the productivity of design systems for the design of nanoelectronic systems. Typically, such a design system consists of software tools (for circuit design), libraries, functional units, methodologies, design processes and of course team capabilities (for the use of these elements). The aim of PRODUKTIV+ is to measure (and finally increase) design system productivity.

Design productivity is influenced by many factors like i.e. the quality of the design, the capability of the designer or the performance of the machine. The measurement of performance of software tools (within the design system) is the focus of this thesis. This means automatic data acquisition of the variables which are relevant for the productivity model.

The objective of this project is to design a data warehouse that contains collected design project data for the purpose of data mining for trends in productivity. A second objective is to define an architecture which is able to extract unformatted data from the tools in the design flow, and transform and load it into the data warehouse.
Preface

Data warehousing is such a huge and exciting area of Computer Science. During my study at the University of Twente, I attended the course “Data Warehousing and Data Mining”, which was a completely new area for me at the time. I found this area rather intriguing, and studied related topics in my own free time. Therefore, when Infineon introduced the project PRODUKTIV+, which involves a lot of data warehousing, I was immediately excited and challenged.

When I started writing my thesis, I discovered the enormous size of the project PRODUKTIV+, whereby my project was a well defined part of it. Although the scope for my thesis was clearly set, it was mandatory to have a good understanding of the overall project. This thesis covers the topics database modeling, data warehousing and ETL processing, (advanced) data extraction from unformatted tool log data, and much more. I’m hopeful that the results presented in this work will be a starting point for the implementation of an operational data gathering architecture, which is able to extract project related log information, and store it in a data store.

Michael Schilling,
München, June 2007
Acknowledgements

I would first like to thank my supervisors from the University of Twente, Maurice van Keulen and Roland Müller, for their valuable input during the last six months. Every time when I travelled to the Netherlands, they managed to find some free time for to set up a meeting. They both really guided me through the research from the beginning to the end. Furthermore, I’d like to give thanks to my colleagues at Infineon, the company I have been writing my thesis at. Infineon has been a very enjoyable place to work. Special mentions have to go to Gerhard Achtstätter, who provided me lots of different design tool log data, which is used to base most of my work on. Another special mention has to go to Jürgen Alt, who made it possible for me to do my final thesis at Infineon. Another special mention goes to Frank Badstübner, who was my supervisor at Infineon. He was always willing to provide help and feedback, something I really appreciate. He also introduced me to the experts in certain fields, so that I could always acquire the essential information in time. As a student, I definitely couldn’t wish for better support. Finally, I want to thank my family and friends. They were always inspiring and motivation, which helped me to complete my thesis in time.
# List of contents

- Abstract ...............................................................................................................................4
- Preface ...................................................................................................................................5
- Acknowledgements ...............................................................................................................6
- List of contents ...................................................................................................................7
- List of figures ......................................................................................................................9
- List of tables .......................................................................................................................10

## 1 Introduction

1.1 Background ..................................................................................................................11
1.2 Project context .............................................................................................................12
1.3 Project analysis ............................................................................................................13
1.4 Research Objectives .................................................................................................14
1.5 Research Questions ....................................................................................................15
1.6 Research method ........................................................................................................15
1.6.1 CRISP-DM ........................................................................................................15
1.6.2 Approach .............................................................................................................17

## 2 Scope and Case

## 3 Literature

3.1 Inheritance strategies for databases .............................................................................24

## 4 Data Sources

4.1 Introduction to the Infrastructure ................................................................................28
4.1.1 InWay .................................................................................................................28
4.1.2 Software Licenses & License Models ................................................................29
4.1.3 Platform LSF ......................................................................................................30
4.2 Parameter Selection .................................................................................................30
4.3 EDA Tool Log Data ..................................................................................................31
4.3.1 Format ................................................................................................................31
4.3.2 File organisation ..................................................................................................34
4.4 Licensing Log Data ....................................................................................................36
4.4.1 Format ................................................................................................................36
4.4.2 File organisation ..................................................................................................36
4.5 Conclusion .................................................................................................................37

## 5 Architecture

5.1 Source Schema ..........................................................................................................39
5.2 Target Schema ..........................................................................................................40
5.3 Extract, Transform and Load (ETL) ..........................................................................42
5.4 ETL Tool .....................................................................................................................45
5.4.1 Relational model .................................................................................................47
5.5 Conclusion .................................................................................................................49

## 6 Database

6.1 Mapping from the Class Model to the Relational Model ...........................................50
6.1.1 Rational Rose UML Data Modeling Profile .......................................................50
6.1.2 Step 1: Model Classes .......................................................................................53
6.1.3 Step 2: Identify persistent objects ......................................................................54
6.1.4 Step 3: Assume each persistent class maps to one relational table .................54
6.1.5 Step 4: Select an inheritance strategy ...............................................................55
12.2 PRODUKTIV+ Class Model (based on ontology version 1.6).................................97
12.3 Relational Data Model I ....................................................................................98
12.4 Relational Data Model II...................................................................................99
12.5 T-HIT Test report ............................................................................................100
12.6 Business Question 1 result .............................................................................102
12.7 Business Question 2 result .............................................................................104

List of figures
Figure 1.1 - Simple Data Gathering Architecture ......................................................13
Figure 1.2 - CRISP-DM data mining project life cycle ...............................................15
Figure 2.1 - Work Packages for PRODUKTIV+ .......................................................19
Figure 2.2 - Data Gathering in more detail ..............................................................21
Figure 2.3 - Thesis Scope .........................................................................................22
Figure 3.1 - Simple Class Hierarchy ........................................................................24
Figure 3.2 - Mapping to a single Table ....................................................................25
Figure 3.3 - Mapping concrete Classes to Tables ....................................................25
Figure 3.4 - Mapping each Class to its own Table ..................................................26
Figure 4.1 - Activity on multiple Data Sources .........................................................28
Figure 4.2 - Snapshot of EDA Tool Log File ............................................................31
Figure 4.3 - Hierarchical overview of Log Files .......................................................32
Figure 4.4 - Structure and Context of a level-1 Log File .........................................34
Figure 4.5 - Design Activity Process Flow ...............................................................35
Figure 4.6 - EDA Log Data Folder Hierarchy ............................................................35
Figure 4.7 - Snapshot of License Log File ................................................................36
Figure 4.8 - License Log Data File Organisation .....................................................37
Figure 5.1 - ETL Concepts and Relationships .........................................................39
Figure 5.2 - Logical Data Model of Target Schema ...............................................40
Figure 5.3 - Context Blocks .....................................................................................41
Figure 5.4 - ETL Architecture ..................................................................................43
Figure 5.5 - Source Component ................................................................................45
Figure 5.6 - RegEx Block Iterator Component .........................................................46
Figure 5.7 - RegEx Parameter Component ...............................................................46
Figure 5.8 - Target Component ................................................................................46
Figure 5.9 - Example of Parameter passing in Data Extractor ..................................47
Figure 5.10 - Data Model of Data Extractor ............................................................48
Figure 6.1 - UML Data Profile - Table display ..........................................................51
Figure 6.2 - UML Data Profile - Table and Columns display ....................................51
Figure 6.3 - UML Data Profile - Table displaying simple behaviour .......................52
Figure 6.4 - UML Data Profile - Table displaying additional behaviour ..................52
Figure 6.5 - UML Data Profile - Table relationships ................................................53
Figure 6.6 - Example of Association Mapping .........................................................56
Figure 6.7 - Example of Many-To-Many Case ..........................................................57
Figure 6.8 - Example of One-To-Many Case .............................................................58
Figure 6.9 - Example of Relationship Roles .............................................................58
Figure 6.10 - ANSI-SPARC Architecture .................................................................59
Figure 6.11 - Differences between Levels in ANSI-SPARC Architecture ...............60
Figure 7.1 - Primary Components of Data Warehouse .............................................62
Figure 7.2 - Dimensionally modelled Data Warehouse ............................................65
List of tables

Table 4.1 - Availability of Parameters ................................................................. 30
Table 4.2 - Comparison between EDA Tool Log Files........................................ 33
Table 5.1 - Source Schemas .................................................................................. 40
Table 6.1 - Mapping Persistent Classes to Relation Tables ................................. 54
Table 6.2 - Analysis Object to Data Model Data Type Mapping ........................ 56
Table 7.1 - Business Questions’ data source dependencies ................................. 65
Table 8.1 - Available parameters in Quick Analysis ............................................. 81
Table 9.1 - Test results: Data Extractor execution times (in seconds) ................... 87
1 Introduction

1.1 Background

In the coming years, nanoelectronics will offer the opportunity to put more and more functionality in increasingly compact space at our disposal for everyday life. Latest developments enable mobile and space saving applications for e.g. automobiles, household and leisure. The high functionality of these applications will be implemented using circuits with feature sizes of 90 nanometres and less. Thus, it will be possible to manufacture more than 100 million transistors in one single device or chip. The coming decade holds large challenges to cost-efficiently develop the functions that became feasible using latest manufacturing technology. This calls for a significant increase in productivity for circuit design.

Generally, this size relation entails the availability problem of necessary design personnel and its costs. Consequentially, this leads to the need for a higher productivity in circuit design. Another motivator for higher productivity is the duration of design projects. Even with today’s team sizes and design expenditures at the upper limit of manageability, a growing number of design projects fail to meet the market entry windows and product cycles dictated by dynamic markets. Due to the pressure to meet these entry windows, many design projects are aborted before their targets were reached. This highlights another problem: today, circuit design productivity cannot be specified with sufficient accuracy.

Ever since the emergence of microelectronics, electronic design automation (EDA) software tools (beside the circuit designers’ qualification) have been the decisive means to increase design productivity. The use of gates for fundamental logical functions as well as digital circuit synthesis were methods that led to marked productivity increases. Today however, there is a growing disparity between the potential offered by manufacturing and the chip design productivity (ITRS roadmap [1]). Obtaining a specific productivity increase requires the capability to determine the quantitative effect of certain modifying actions ahead of time. Knowing the productivity of circuit design is the key to calculating the business economies of EDA investments that serve to increase design capabilities.

Until now, the quantitative assessment of designed or planned tool or methodology improvements is still unsolved. The semiconductor industry does not have an assessment model or methodology to determine design productivity that is precise enough to provide the desired objective foundation for decisions in the EDA area. The project at hand intends to develop the means for the necessary quantitative measurement and pro-active estimation of productivity. To do so, the project PRODUKTIV+ consortium has the following goal:

*Modeling and measurement of design productivity in the semiconductor industry*

The project PRODUKTIV+ is funded by the German government, and is an initiative of several parties:
Infineon has been working on the subject of micro- and nanoelectronics design systems for several years. A result of this work is the in-house design system InWay, which comprises a large number of software tools and libraries that support a wide spectrum of design tasks (digital, mixed signal, analogue). The integrated libraries provide access to Infineon’s high-end manufacturing technologies for CMOS logic products. Almost all integrated software tools were delivered by external vendors.

The InWay design system is available world-wide at Infineon sites in Germany, Europe, Asia and North America. Apart from a unified distribution and maintenance, there is a global license management for cost-optimized license deployment for distributed design teams [2]. More than 1500 internal design engineers worldwide and several external cooperation partners use the system. The actual design system and the distribution, maintenance and license management infrastructure which is available in parallel to the system, will form an important base for the activities in PRODUKTIV+. Due to the large number of users, data volumes with good statistical accuracy can be expected for the intended productivity measurements. The performance of the design team in conjunction with the utilization of the design system was measured using the Numetrics system [3].

1.2 Project context

The project PRODUKTIV+ deals with the productivity of design systems for the design of nanoelectronic systems. Typically, such a design system consists of the following elements:

- EDA software tools for circuit design and their synchronized delivery within a design flow (e.g. system design, RTL-to-GDS design).
- Libraries of standard elements for fundamental logic functions, embedded memory and input/output blocks.
- Prefabricated, reusable and complex functional units (intellectual property – IP).
- Methodologies and design processes which are made available to the designer for certain design tasks (e.g. low power design, analogue design).
- Team capabilities (e.g. education, experience) for the use of the above elements.

In the area of nanoelectronic systems design, the term ‘productivity’ is often used as a scoring metric for project success, but up to now, this variable cannot be measured
satisfactorily. Consequently, a reasonably accurate project and capacity planning does not exist in semiconductor design.

PRODUKTIV+ intends to provide modeling, measurement and evaluation of the productivity aspects of all components of a design system as well as to set up a link to economic index schemes. For this purpose, the research activities are planned on the following interrelated aspects:

- Development of a productivity model that covers the above components of the design system,
- Automatic data acquisition of the variables (parameters) which are relevant for the productivity model,
- Computer-based analysis and simulation techniques, and
- An exemplary application of research results for various design projects that cover different requirement profiles for circuit design.

The overall interest for PRODUKTIV+ is design system project productivity. The productivity of a design system project is influenced by many factors like e.g. the quality of the design, the capability of the designers or the performance of the machine.

What is actually going on during the design system project? This is an easy question to ask, but rather hard to answer. If you want to observe designers, you could interview designers and ask what they are doing. The first reason why this approach will fail, is because people might not always want to cooperate. Secondly, since you are dealing with people, the results are subjective. A third reason why this is not a solution, is because designers are spread all over the world, what makes it hard to retrieve a complete set of data. Therefore there is an urgent need for a way to observe the process by collecting tool log data in an automatic manner. This is the main focus of this thesis, setting up an architecture for automatic data acquisition. It is also important to realise that Infineon intends to measure machine data, not people.

### 1.3 Project analysis

Automatic data acquisition involves retrieving relevant information from software tools during (or after) the design process. When software tools are executed they produce log files (tool output). These log files are unformatted files which contain information of the execution. Extracting data from these log files comes with many challenges.

One constraint in this project, is to store extracted log data in an intermediate file (XML format). The reason for this XML file, is to have a transparent format for all partners to do analysis on. After data extraction, it is loaded into a data store (e.g. database or data warehouse). This process is depicted in Figure 1.1.

![Figure 1.1 - Simple Data Gathering Architecture](image-url)
Some problems are the following:

- There is lack of real world data: since there is no architecture for automatic data gathering yet, only few data is available.
- There is lack of EDA tool evaluations: there is no fixed solution to measure and evaluate tool performance.

EDA tools produce log data, but it is unclear when log files are created, and were they are located. This is one important task to find out. Also, task executions of a tool might have multiple iterations, whereby it is unknown whether log files are overwritten, appended or whether new files are created. Therefore it is important to investigate the lifetime of log files.

Besides log data from EDA tools, there are other sources to investigate. For instance log data from a licensing server might help to discover what happened during an activity. It is important to discover what data sources contribute to a better understanding of the design process, and how to relate one with another.

In this thesis there are two important business questions to ask:

BQ1: For how long did tool X use license Y during project Z?

BQ2: What is the total CPU usage of tool X, during project Z?

Chapter 2 describes the scope and case of this project in more detail.

1.4 **Research Objectives**

The first objective of this project is to define an ETL (for Extract, Transform and Loading) architecture which is able to extract data from different sources in the design flow and transform and load it into a database or data warehouse. Extracting data from unformatted EDA tool log data is the key challenge here.

The second objective is to define a database that contains PRODUKTIV+ ontology related data. Therefore, the data model is led by the PRODUKTIV+ ontology. An ontology is a data model (or knowledge base) that represents a set of concepts within a domain and the relationships between those concepts\(^1\). The database provides a set of dimensional tables (input) for the data warehouse, the following objective. The database is necessary to store partner *A-box* data, that will be used in the next phase of PRODUKTIV+. The term *A-box* is explained in the next chapter.

The third objective is to design a data warehouse that contains collected design project data for the purpose of data mining for trends in productivity. This data warehouse should be feasible to answer the two main business questions. The data warehouse is necessary to store extracted log data.

\(^1\) [http://en.wikipedia.org/wiki/Ontology_(computer_science)]
1.5 Research Questions

There are several interesting questions that can be asked. The research questions are organized per objective.

Objective 1: architecture design:
- What is the lifetime of a data source?
- Where can we find data sources?
- What is the structure of EDA tool log files?
- What are the ETL processing steps?

Objective 2: database design:
- What does the ontology look like?
- What does the data model look like?
- What information do we need to store?

Objective 3: data warehouse design:
- What does the data warehouse look like?
- What data do we store in the data warehouse?
- What data sources are available?

1.6 Research method

1.6.1 CRISP-DM

CRISP-DM is followed for guidance during this research. CRISP-DM is a methodology for an industry- and tool-neutral data mining process model [4]. This process is shown in Figure 1.2.

Figure 1.2 - CRISP-DM data mining project life cycle
The life cycle of a data mining project consists of six phases. The sequence of the phases is not strict. Moving back and forth between different phases is always required. It depends on the outcome of each phase which phase, or which particular task of a phase, that has to be performed next. The arrows indicate the most important and frequent dependencies between phases.

The outer circle in the figure symbolizes the cyclic nature of data mining itself. A data mining process continues after a solution has been deployed. The lessons learned during the process can trigger new, often more focused business questions. Subsequent data mining processes will benefit from the experiences of previous ones.

Since this is not a typical data mining project, not all the activities and phases are relevant. However, CRISP-DM is a powerful methodology because it is well defined, and it addresses every important part of a research. In a following phase of PRODUKTIV+, it deals with data mining and data analysis, and therefore it makes perfect sense to apply this methodology from the start.

Below follows a brief outline of the phases:

**Business Understanding**

This initial phase focuses on understanding the project objectives and requirements from a business perspective, and then converting this knowledge into a problem definition, and a preliminary plan designed to achieve the objectives.

The first task in this phase is to determine business objectives. It is important to find out what Infineon really wants to accomplish. This task outputs the project background, business objectives and the business success criteria.

A second important task is assess situation. This task involves more detailed fact-finding about all of the resources, constraints, assumptions and other factors that should be considered in determining the data analysis goal and project plan.

Finally, a project plan is produced. This plan describes the intended plan for achieving the business goals.

**Data Understanding**

The data understanding phase starts with an initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information.

There is a large collection of data available from an internal test case called *T-HIT*. Compared to a typical design project, *T-HIT* is rather small. Where other projects take weeks or even months to complete, *T-HIT* runs in only a few hours. Nevertheless, the data collection of *T-HIT* is probably more complete than any other design project can deliver at the moment. Therefore it forms a good starting point to describe the data.
Two important tasks in this phase are data quality verification and data exploration. Data exploration tackles the main business questions, which can be addressed using querying, visualization and reporting. Data is inspected to discover relationships between attributes. To verify the data quality, the quality of the data is examined. Here it becomes clear whether the data is complete or correct.

**Data Preparation**

The data preparation phase covers all activities to construct the final dataset (data that will be fed into the modeling tool(s)) from the initial raw data. Data preparation tasks are likely to be performed multiple times, and not in any prescribed order. Tasks include table, record, and attribute selection as well as transformation and cleaning of data for modeling tools.

In this phase all so called ETL (Extract, Transform and Load) processing steps are defined. This involves e.g. selecting, cleaning, formatting and integrating data. The output of this phase is an important output for this project, namely a large data set with a collection of extracted design project data. This data set forms the input for the next phase of PRODUKTIV+, where data analysis and data mining is applied.

**Modeling, Evaluation, Deployment**

The phase modeling, evaluation and deployment are less important in the scope of this thesis. Of course modeling and evaluation are two important parts of this research, but in the context of CRISM these have a different meaning (more data mining oriented).

### 1.6.2 Approach

The approach to make the goal of measuring tool productivity, several preliminary tasks should be performed:

- Literature study
- Architecture design
- Database design
- Data warehouse design
- Prototype
- Evaluation

The research starts with the literature study. The literature to study, depends on the expected topics in this project. Different modeling techniques have to be studied for the database and data warehouse design. Important keywords for the architecture are *ETL processing* and *data warehousing*. Both are explained in more detail later in this thesis.

Designing the architecture is an activity that can start, after the business understanding phase is completed. At this point it must be clear what kind of infrastructure you are dealing with, and what the important data sources are. The key activity in this task, is to identify important data sources, to have general knowledge about how to process this data, and to finally store this data in a data store, everything in an automatic manner. Different techniques (e.g. *XSLT, SQL*) are studied to perform these ETL steps. One of the important aspects for the architecture, is that the architecture should be flexible enough to deal with future changes. The variety of log data is unlimited, and therefore the design
requires a flexible solution for expressing extraction rules, to be able to extract information from log data.

The database is designed in such a way that it is led by the PRODUKTIV+ ontology. This is a project constraint, because this database will be an output to the next stage of PRODUKTIV+. The ontology consists of eight different diagrams, which all together cover the area of chip design. Having a good understanding of the ontology, and its concepts, is therefore a necessary skill to acquire in this project. The ontology contains a set of concepts that uses the concept of inheritance. This is by default not supported in relational databases, and therefore it is a topic for the literature study.

There are a few different techniques to model a data warehouse, e.g. ER-modeling and dimensionally modeling. Modeling the data warehouse is one of the activities during the literature study. Depending on the business needs and project constraints, the best technique is selected. During the business understanding phase, most of the knowledge is retrieved to be able to design the data warehouse.

The database, data warehouse and architecture are implemented in a prototype. The architecture will provide basic functionality to extract information from log data. The database will consist of data that can be extracted automatically (e.g. design project info, tool information, license types) and data that cannot (e.g. tool activities and design parameters). The latter has to be added either manually, or retrieved from worksheets or other (digital) documents.

The data warehouse prototype will contain historical data from the design process. License usage and CPU usage are the two main facts, which allows answering both business questions. Since all information cannot be retrieved from one single data source, different attributes have to be merged. For the data warehouse, the database will be the most important source to relate data.

The final part of the project is evaluation. All key parts of the research are evaluated, based on different criteria. The architecture for automatic data extraction is the key topic of this research, and the most important criteria for this architecture is reliability. The extracted information should be reliable, or else it would give a totally different meaning to the measurement of (tool) productivity.

The test design T-Hit is set up in such a way that it produces a report summary at the end of the run. This report summary shows all project steps, and additional information of each single step (e.g. duration, exit status). This test report will be an important data set to verify the data extraction results, and proof that the extraction itself is reliable.
2 Scope and Case

This chapter describes the scope and case of the project in more detail. The first part explains PRODUKTIV+ and the motivation for this. The second part zooms into the scope of this thesis, which is a well defined part of PRODUKTIV+.

PRODUKTIV+ is divided into four phases, so called work packages. Each work package contains primary tasks and activities. These four work packages are depicted in Figure 2.1.

![Figure 2.1 - Work Packages for PRODUKTIV+](image)

Here follows a brief overview of the scientific goals of each work package:

**Modeling**

For an application of the general productivity formula to a design system, the values of input data (personnel, time, and materials) can be recorded. This is relatively straightforward and can be done in analogy to concepts used in the production field. Quantifying the output of design activities is harder. As part of the project, viable, comparable metrics for the output of a design system must be defined. The investigation of factors that influence the productivity graph of a design system is another crucial goal of the project. Integration with economic index schemes is planned, so that the productivity model can contribute to the most relevant economical aspects of decision-making for EDA investments. This integration will for the first time allow quantifying a return-on-investment in the area of EDA investments. To obtain this integration, suitable coupling variables and interaction interfaces must be identified in cooperation with organisations that are responsible for business management.

**Data Gathering**

In order to define the parameters of the productivity model, the model needs to be verified using real design project data. To do so, the influencing factors of the productivity model must be described in a generally valid data model, and a system architecture with interfaces for data gathering and analysis needs to be defined. Another
goal is to automate data gathering in the cooperating semiconductor companies using a suitable link to existing design systems so that the expenditures for productivity measurements can be minimized.

PRODUKTIV+ clearly does not intend to assess the design teams’ or its members’ capabilities. The goal of the ‘data gathering’ project section is to enable an automatic gathering of large data volumes. These data will be acquired in the course of a design project.

**Data Analysis and Simulation**

The research goal in the area of analysis techniques is to identify the relevant variables influencing the productivity of individual components and of the components in mutual interaction for any given design system. This knowledge permits to define specific measures that can be used to shift the productivity graph towards higher output. Research activities on the subject of simulation techniques focus on gaining quantitative productivity forecasts for changes to the design system and for design projects which use that design system. Since economically relevant index numbers are linked with the productivity model, the economic effects of investments in the design system can be predicted beforehand, providing a quantitative base for the decision process.

**Integration and Assessment**

The research findings need to be applied to example design projects with different requirement profiles (analogue, digital, various application areas). To do so, they will be integrated and linked as a reference system with the corresponding design systems in the cooperating semiconductor companies. The purpose of this linkage is to assess the accuracy of the productivity model using data from real design situations and to maximize it by optimizing its parameters. At the same time, this should deliver a proof of the viability of automated data acquisition and support preparations for a further utilization of the results.

PRODUKTIV+ started on October 1st 2005, and has a total duration of three years. At the start of this thesis, work package one (‘Modeling’) was completed. Therefore, the focus in this thesis is work package two, ‘Data Gathering’. Figure 2.2 shows this work package in more detail. It shows the four work packages, whereby ‘Data Gathering’ is shown in more detail. It contains four main components, whereby the abstract description is shown on the left side. An example is shown on the right side of each component.

The terms **T-Box** and **A-Box** are used to describe two different types of statements in **ontologies**. An **ontology** is a data model (or knowledge base) that represents a set of concepts within a domain and the relationships between those concepts. The language **OWL** (for Web Ontology Language) [5] is used to develop (and describe) the **ontology**.

The **T-Box** is a **terminological component**, a vocabulary associated with a set of facts **A-Box**. **T-Box** statements describe a system in terms of controlled vocabularies, for example, a set of classes and properties.
An *A-box* is an *assertion component*, a fact associated with a terminological vocabulary within a knowledge base. *A-Box* are *T-Box*-compliant statements about that vocabulary.

*T-Box* statements tend to be more definitional in nature such as a dictionary of words. *T-Box* statements are sometimes associated with object-oriented classes and *A-Box* statements associated with instances of those classes. Together *A-Box* and *T-Box* statements make up a knowledge base.

```
{ ..., GenericTask, GenericActivity, Resource, ... }
```

```
{ ..., Synthesis, Simulation, Verification, ... }
```

**Figure 2.2 - Data Gathering in more detail**

The two upper components are shared among partners. On this abstraction level, there is a high degree of similarity for all partners. The PRODUKTIV+ *T-Box* contains the schema of this *ontology*. It implements the abstract concepts (classes), e.g. a tool, resource or license. The PRODUKTIV+ shared *A-Box* describes the data on such a level that it is valid for all partners. There are three activities show in the example.

The two bottom components are partner specific and contain more concrete data. Each partner can have one partner *A-Box* and several project *A-Box* components. Because of the sensitivity of this information, e.g. price information and license costs, this information is company confidential and therefore only visible for the company itself. Both *A-box* components add more meaning to the data and the granularity increases. In the first example, it shows more detailed information regarding a simulation. In the last example it also contains concrete price information.
The PRODUKTIV+ ontology is an outcome of the first work package. It models all typical events and entities within the domain of ‘chip design’. This ontology is an important input for this thesis, since this is the language all partners speak and agreed on. It happens to be there, and decisions and solutions in this project have to be compliant with it. Figure 2.3 shows a clear overview of the scope of this thesis.

![Figure 2.3 - Thesis Scope](image)

The scope of the thesis is indicated with a dotted line. It shows that, although the T-Box is not in the scope, the ontology is an important input. Log data are sets of data, produced by a data source (e.g. a license server). This overview also contain a database (P+) and a data warehouse (DW), which will form an important input for the next work packages ‘Data Analysis’ and ‘Simulation’. The two large arrows show where a motivation arises, which is explained in more detail later on.

Within the scope of this thesis are the following:
- Architecture for data gathering
- Data Sources (Log data)
- Database design (P+)
- Data warehouse design (DW)

Data Sources

This project deals with data extraction from multiple data sources. Chapter 4 describes several data sources, and the necessity to extract from multiple sources, instead of extraction from a single source. This chapter starts with an introduction to the system,
were these data sources are part of. This is followed by a more detailed description of the selected data sources and a format description of the log data it produces.

**Architecture**

Automatic data gathering comes with many challenges. One of the main challenges is, to design an architecture that is flexible enough to do deal with any type of log data. A second challenge is, that the architecture should be manageable. This means that different users, with average knowledge, should be able to use the system. This topic is described in chapter 5.

**Database design**

In the work package to follow, multiple instances of A-Boxes are created, and filled with real data. This data has to come from somewhere, what is the motivation for the database design. It is led by the ontology, and will therefore contain a subset of the ontology’s concepts. The database design is described in chapter 6.

**Data warehouse design**

The next work package also requires the creation of multiple A-Box instances, and filled with project data. This data is extracted form log data and loaded into a data warehouse. The main goal for the data warehouse is to collect data for answering both business questions. The data warehouse design is described in more detail in chapter 7.
3 Literature

This chapter describes inheritance structures for databases. This is one of the topics that is covered during the research study. Besides this topic, two other important topics are studied, namely:

- XML Modeling
- Data warehousing and ETL processing

Both topics are not covered in this chapter. The first topic, ‘XML Modeling’, is focussed on mapping techniques, for mapping XML to a (relational) database[9][10][11]. This topic is important to research, because the architecture deals with a large collection of XML documents that have to be loaded into a database. The second topic is focused on data warehousing and ETL processing [12][13] and conceptually modeling it [6][7].

3.1 Inheritance strategies for databases

Relational databases do not natively support inheritance. Therefore you have to map the inheritance structures within your object schema to your data schema. This section describes three different scenarios how to deal with inheritance.

Consider the following class hierarchy:

```
    Indicator
        value

       PowerIndicator
            tempRangeFrom
            tempRangeTo

       AreaIndicator
            dieArea
            totalWireLenght
```

Figure 3.1 - Simple Class Hierarchy

Figure 3.1 displays a class hierarchy whereby Indicator is a superclass, and PowerIndicator and AreaIndicator are two subclasses. This example is used to explain the three different inheritance strategies in this section.

Scenario 1: Map hierarchy to a single table

See Figure 3.2. Two columns have been added to the table, IndicatorID and IndicatorType. The first column is the primary key for the table, indicated by the <<PK>> stereotype, and the second is a code indicating whether the indicator is a power indicator, an area indicator, or perhaps both.
Setting up a data gathering infrastructure for PRODUKTIV+

### Indicator

<table>
<thead>
<tr>
<th>IndicatorID</th>
<th>IndicatorType</th>
<th>Value</th>
<th>TempRangeFrom</th>
<th>TempRangeTo</th>
<th>DieArea</th>
<th>TotalWireLenght</th>
</tr>
</thead>
</table>

**Figure 3.2 - Mapping to a single Table**

The `IndicatorType` column is required to identify the type of object that can be instantiated from a given row. For example, the value of ‘P’ would indicate the indicator is a power indicator, ‘A’ would indicate an area indicator, and ‘B’ would indicate both. Of course both, meaning the indicator is a power indicator and an area indicator, does not make sense in this example.

Although this approach is straightforward it tends to break down as the number of types and combinations begin to grow. For example, when you add the concept of a flexibility indicator you need to add a code value, perhaps ‘F’, to represent this. Now the value of ‘B’, representing both, is no longer meaningful (or valid). Furthermore you might have combinations involving flexibility indicators now, so you would need a code for this.

### Scenario 2: Map each concrete class to its own table

**PowerIndicator**

<table>
<thead>
<tr>
<th>PowerIndicatorID</th>
<th>Value</th>
<th>TempRangeFrom</th>
<th>TempRangeTo</th>
</tr>
</thead>
</table>

**AreaIndicator**

<table>
<thead>
<tr>
<th>AreaIndicatorID</th>
<th>Value</th>
<th>DieArea</th>
<th>TotalWireLenght</th>
</tr>
</thead>
</table>

**Figure 3.3 - Mapping concrete Classes to Tables**

With this approach a table is created for each concrete class, each table including both the attributes implemented by the class and its inherited attributes, see Figure 3.3. There are tables corresponding to each of the `PowerIndicator` and `AreaIndicator` classes because they are concrete, objects are instantiated from them, but not `Indicator` because it is abstract. Each table was assigned its own primary key, `PowerIndicatorID` and `AreaIndicatorID` respectively. To support the addition of `FlexibilityIndicator` all you need to do is add a corresponding table with all of the attributes required by `FlexibilityIndicator` objects.
Scenario 3: Map each class to its own table

Following this strategy you create one table per class, with one column per business attributes and any necessary identification information (see Figure 3.4). The data for the `PowerIndicator` class is stored in two tables, `Indicator` and `PowerIndicator`, therefore to retrieve this data you would need to join the two tables (or do two separate reads, one to each table).

![Figure 3.4 - Mapping each Class to its own Table](image)

Each derived class table in the database has a one-to-one relationship with the base class table along with an existence dependency (meaning the derived class table cannot have a row unless there is a corresponding row in the base class table).

Notice how `IndicatorID` is used as the primary key for all of the tables. For the `PowerIndicator` and `AreaIndicator` tables the `IndicatorID` is both a primary key and a foreign key. In the case of `PowerIndicator`, `IndicatorID` is its primary key and a foreign key used to maintain the relationship to the `Indicator` table. This is indicated by application of two stereotypes, `<PK>` and `<FK>`.

Comparing different approaches

These three different approaches are compared in appendix 12.1. The first strategy, one table per hierarchy, is quite easy to implement. To add new classes, you just need to add columns for the additional data. One drawback of this approach, you waste a lot of potential space. Use this strategy for simple and/or shallow class hierarchies where there is little or no overlap between the types within the hierarchy.

The second strategy, one table per concrete class, is especially useful for ad-hoc reporting because all the data you need about a single class is stored in only one table. However, when you modify a class you need to modify its table and the table of any of its subclasses. Therefore this strategy is recommended when changing types and/or overlap between types is rare.

The last strategy, one table per class, is easy to understand because of the one-to-one mapping. This strategy is the simplest and most flexible inheritance mapping, where each class in the inheritance hierarchy (base or derived) is mapped to its own table in the
database. This strategy allows a huge flexibility when changing and/or adding superclasses. One small drawback of this approach is the amount of tables you need (and have to maintain). When there is significant overlap between types or when changing types is common, this strategy is preferred.
4 Data Sources

What is really going on during the design of a project? Unfortunately this is a hard question to answer. Standing at the designers’ desk and observe might lead to some good (correct) results, but is rarely appreciated by the person itself. Handing over questionnaires might also lead to good results, but is too subjectively. Therefore we want to observe ‘what is going on’ by extracting (and analyzing) data from log files.

Figure 4.1 shows the presence of a certain activity, on three different data sources, over time.

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>8 AM - 5 PM</td>
</tr>
<tr>
<td>Versioning</td>
<td>10 AM - 4 PM</td>
</tr>
<tr>
<td>LSF</td>
<td>2 PM - 4 PM</td>
</tr>
</tbody>
</table>

Figure 4.1 - Activity on multiple Data Sources

From the first data source, License, one can tell that a tool license is checked out for a period of time. Typically, a license enables a tool to run. The second data source, Versioning, holds versioning information. Versioning enables multiple users to work on the same design. In the figure, this data sources shows that a part of the design is checked out for a period. The third data source, LSF, shows the occurrence of a tool execution (i.e. a simulation) during the interval.

Relying on a single data source is insufficient, because context information is missing (e.g. project- and design details). It clearly does not tell what the designer is doing during these activities. For instance, typically the designer spends time to prepare and setup the project. Also, the designer spends time ‘thinkin’. This information is important to know, because it might also contribute to the measurement of ‘productivity’. Nevertheless, the initial goal is to derive information from tool activities via log data. By merging data from different data sources, it becomes possible to measure ‘what is going on’ (to some extent). However, the activities shown in the example are (measurable) tool activities. This chapter describes the most useful and reliable data sources for data extraction.

4.1 Introduction to the Infrastructure

4.1.1 InWay

Electronic Design Automation (EDA) is the category of tools for designing electronic systems ranging from printed circuit boards (PCBs) to integrated circuits. The term EDA is also used as an umbrella term for computer-aided engineering, computer-aided design and computer-aided manufacturing of electronics in the discipline of electrical engineering. This thesis describes EDA specifically for electronics, and concentrates on EDA used for designing integrated circuits. The segment of the industry that must use
EDA are chip designers at semiconductor companies. Large chips are too complex to design by hand.

The InWay design system is Infineon’s name for an integrated set of several design flows for IC-design and multiple design packages for Infineon’s technologies. A design flow is a set of carefully selected EDA tools, additional software and scripts to make these tools work together. To cope with various design requirements different design flows are necessary.

The design system is the design flow plus packages (libraries). The design flow integrates approximately 30 tools, mainly from the EDA vendors Cadence, Magma, Mentor and Synopsys. One important feature of it is Global License Management (GLM), which manages tool license usage. This is explained in more details in the next paragraph.

Design packages are the essential building blocks for a chip design. A design system is only functional in case of at least one design package integrated. A design package contains Libraries. A library is a set of building blocks for a chip, e.g. standard cells, memories, cores, i/o interfaces. Views are different representations of silicon behaviour of the building blocks.

The connection between a design package and the design flow is a defined interface (library-, technology views). The flow development projects and the design package development projects share the responsibility for the definition of this interface, what is called the view specifications.

4.1.2 Software Licenses & License Models

A software license comprises the permissions, rights and restrictions imposed on software (whether a component or a free-standing program). Use of software without a license could constitute infringement of the owner’s exclusive rights under copyright or, occasionally, patent law and allow the owner to sue the infringer.

Under a software license, the licensee is permitted to use the licensed software in compliance with the specific terms of the license. If there is a breach of the license, depending on the license it may result in termination of the license, and potentially the right of the owner to sue.

A software vendor may offer a software license unilaterally (without giving the licensee the opportunity to negotiate for more favourable terms) such as in a shrink wrap contract, or even as part of a software license agreement with another party. Virtually all mass produced proprietary software is sold under some form or fashion of software license agreement. One off, or custom software is often licensed under terms of which are specifically negotiated between the licensee and licensor.

In addition to granting rights and imposing restrictions on use of the software, software licenses typically contain provisions which allocate liability and responsibility between the parties. In enterprise and commercial software transactions these terms (such as limitations of liability, warranties and warranty disclaimers, and indemnity if the software infringes intellectual property rights of others) are often negotiated by attorneys specialized in software licensing. The legal field has seen the growth of this specialized
practice area due to unique legal issues with software licenses, and the desire of software companies to protect assets which, if licensed improperly, could diminish their value.

For larger companies, these licenses are worldwide arranged and enabled using a license-manager, FLEXlm2. FLEXlm is the most popular license manager used in the software industry. FLEXlm is best known for its ability to allow software licenses to be available (or float) anywhere on a network, instead of being tied to specific machines. Floating licensing benefits both users and license administrators. Users make more efficient use of fewer licenses by sharing them on the network. License administrators control who uses the licensed application and the machine(s) where the licenses are available. The name for Infineon’s implementation of the system is Global Licensing Management (GLM).

### 4.1.3 Platform LSF

*Platform LSF* (Load Sharing Facility) is software for managing and accelerating batch workload processing for compute-and data-intensive applications. With *Platform LSF*, you can intelligently schedule and guarantee completion of batch workload across a distributed, virtualized IT environment regardless of operating system, including desktops, servers and mainframes.

A designers’ workflow cycle involves spending time thinking, editing text files, and other activities that does not require high computation power. The general idea of the LSF environment, is to have a shared compute farm available when high computation power is required. So while designer A spends time editing text file, designer B uses LSF for complicated tasks, for instance to run a simulation. This reduces costs, because in this setup, not every designers’ desk have to be equipped with expensive machines.

### 4.2 Parameter Selection

Table 4.1 shows a list of parameters to extract, and the availability of it in different data sources.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Tool log (level 1)</th>
<th>Tool log (level 2)</th>
<th>Tool log (level 3)</th>
<th>License log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution start</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Execution stop</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Exit code</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CPU Usage</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Driver start</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Driver stop</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>License checkout</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>License checkin</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.1 - Availability of Parameters

**Execution start**

This is the start time of a tool execution.

---

Execution stop
This is the end time of a tool execution.

Exit code
The exit code shows the status of the process, after the execution has finished.

CPU Usage
This is the total CPU time (in seconds) spent, for a task to complete. Log files only contain CPU usage for LSF jobs.

Driver start
This is a timestamp of a (tool) driver, when loaded.

Driver stop
This is a timestamp of a (tool) driver, when unloaded.

License checkout
This is the timestamp of a license (feature), when checked out.

License checkin
This is the timestamp of a license (feature), when checked in.

4.3 EDA Tool Log Data

4.3.1 Format
EDA Tool log data is a set of log files produced by EDA Tools. These log files are ASCII text files containing log information. Figure 4.2 shows a snapshot from a log file.

Figure 4.2 - Snapshot of EDA Tool Log File
The format of a log file is not fixed. Every tool vendor is free to produce a log output in a desired format of their own. As a consequence, the variety of different formats is unlimited and can change over time. Different formats are also caused by different run options, specified by the designer, to report less or more details.

Different tools are necessary to complete a design. These tools are configured in an InWay project, specified by the designer. A project exists of a number of activities (i.e. place and route). Each activity contains one or more tasks to execute. Activities and tasks are executed in a predefined sequence, specified in the project file. In the simplest scenario these executions are all serial. During this process log data is produced.

To be able to distinguish one log file from another, these files are divided in abstraction levels. Each log file is classified as a level-1, level-2 or level-3 log file. This is depicted in Figure 4.3. Arrows between the log files represent a reference occurrence from one log file to another.

A level-3 log file contains detail information of a single task execution. For instance when a command `bind` is executed, it creates a log file with the name ‘bind.log’. In this abstraction level, the mutual difference between the formats of log files is the highest. Each task itself is quit specific, and therefore produces a huge amount of detail information. The context of this log file is the executed task. Regarding the scope of the contained information this means that it, at the max, belongs to one single task.

![Figure 4.3 - Hierarchical overview of Log Files](image-url)
**Level-2** log files are a summary of all tasks within an activity. Therefore the context of the information is the activity itself. The amount of information in these log files is limited, compared to **level-3** log files. During the design process, data in **level-2** files are appended, while **level-3** log data is overwritten (on a re-run).

**Level-1** log files contain information of the complete run, including all activities and tasks. The degree of detail is much higher compared to **level-2** log files. These log files are also more complete than both other levels, because it contains information about the full design. One major drawback of the **level-1** log file, is that it is only created after completion of the project.

These three levels are compared in Table 4.2. The goal to extract information from log file depends on two important factors. First, it depends on the availability of the desired information (as described in paragraph 4.2. Secondly, this depends on the characteristics of a log file. For instance, when the same information occurs in different files, the lifetime of a file might play an important role.

<table>
<thead>
<tr>
<th></th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Degree of detail</strong></td>
<td>High</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>File creation</strong></td>
<td>After complete project run</td>
<td>After tool completion</td>
<td>Instantly</td>
</tr>
<tr>
<td><strong>Variety in structure for log files off same level</strong></td>
<td>very low</td>
<td>low</td>
<td>very high</td>
</tr>
<tr>
<td><strong>Overwritten on re-runs</strong></td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 4.2 - Comparison between EDA Tool Log Files

Regarding the abstraction levels of log files, **level-1** log files are the main focus in this thesis. Because of the completeness of information, these log files are preferred over the others. **Level-1** log files have a basic structure, which is always the same. It basically consists of **sections** and **subsections** (whereby a **subsection** always belongs to only one **section**). A **section** is a direct result of an activity. As mention before, an activity consists of tasks, so therefore a **subsection** is a result of a single task. This is depicted in Figure 4.4. The terminology from a log file (i.e. **section**, **subsection**) is adopted in this thesis.

These log files also contain driver information, i.e. driver name, start time and end time. In this context, driver means a tool driver. Before a tool can execute tasks, the driver of the tool has to be loaded. So in other words, the driver enables the tool to start and run. The scope of a driver is the activity, and therefore in the log file it belongs to a **section**. It is also possible that nested driver log information is found in the log file. This occurs when, during a task, a second tool is loaded (and executed).
LSF log data is also written to EDA tool log files. Whenever a task is submitted to a LSF queue, a line with the submitted LSF JobID is written to the log file. On interval, InWay queries the LSF server to find out whether there are new LSF results. If there are new results, these are retrieved and written to the log file in the following tabular form:

<table>
<thead>
<tr>
<th>JOB</th>
<th>HOST</th>
<th>TYPE</th>
<th>OSREL</th>
<th>MODEL</th>
<th>CPUF</th>
<th>CPU (sec)</th>
<th>MEM (MB)</th>
<th>SWAP (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>host01</td>
<td>SUNSOL</td>
<td>50</td>
<td>MODEL1</td>
<td>60</td>
<td>280</td>
<td>1000</td>
<td>1010</td>
</tr>
<tr>
<td>8001</td>
<td>host02</td>
<td>SUNSOL</td>
<td>60</td>
<td>MODEL2</td>
<td>80</td>
<td>160</td>
<td>2000</td>
<td>2010</td>
</tr>
</tbody>
</table>

There are three important attributes to mention here. Attribute CPU is important, because it contains the total CPU time (in seconds) of the executed LSF job. Attribute CPUF stands for CPU factor. This value becomes interesting, when comparing LSF results from different hosts. It allows the CPU value to be normalized. The last important attribute to mention is JOB. It is the unique identifier of an executed LSF Job. It enables the LSF result to be related to a certain context (i.e. tool or task). The other attributes are not unimportant at this point.

### 4.3.2 File organisation

Log files are the result of tasks and activities during the project, whereby creation happens on different levels, and therefore create different type of log files (over time). It also depends on the project settings whether a task or activity produces a log file. This is depicted in Figure 4.5.
The infrastructure is organized in such a way, that each activity has an own workspace in the folder hierarchy (Figure 4.6).

Each folder contains the results of a single activity. If applicable, it contains the detailed log file of each task in of the activity. The file ‘Summary.log’ contains a summary of log information of all executed tasks. Sometimes a task or activity is repeated multiple times. As a consequence, log files are overwritten or appended with information. This requires a strategy to extract information in a reliable manner.
4.4 Licensing Log Data

4.4.1 Format

License log files are generated by license servers, as mentioned earlier in this chapter. Information like timestamp of the checkout time, timestamp of the checkin time, duration, user… are stored in this file. See Figure 18 for a sample output.

<table>
<thead>
<tr>
<th>Field 1</th>
<th>Field 2</th>
<th>Field 3</th>
<th>Field 4</th>
<th>Field 5</th>
<th>Field 6</th>
<th>Field 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAST_VERY</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User1</td>
<td>Project1/ftp://106.082.230344000</td>
</tr>
<tr>
<td>BLAST_SPEED</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User1</td>
<td>Project1/ftp://106.082.230344000</td>
</tr>
<tr>
<td>BLAST_BUILDER</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User1</td>
<td>Project1/ftp://106.082.230344000</td>
</tr>
<tr>
<td>BLAST_PLAN</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User2</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_PLANPRO</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User2</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_CREATE</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User1</td>
<td>Project1/ftp://106.082.230344000</td>
</tr>
<tr>
<td>BLAST_CREATE_SA</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User1</td>
<td>Project1/ftp://106.082.230344000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>BLAST_TEST</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User3</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_FOOTSix_REPLACE</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User3</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_VERY</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User3</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_VERY</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User3</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_VERY</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User3</td>
<td>UNKN</td>
</tr>
<tr>
<td>BLAST_SPEED</td>
<td>22.08.2006 12:48:22</td>
<td>22.08.2006 12:48:22</td>
<td>00:00:02</td>
<td>00:00:02</td>
<td>User3</td>
<td>UNKN</td>
</tr>
</tbody>
</table>

**Figure 4.7 - Snapshot of License Log File**

Field 1 is the name of the checked out license feature. A license feature is associated with a tool license. Usually, tool licenses have multiple license features. In several cost models, license costs are different for each license feature.

Field 2 and Field 3 hold time information of the activity. The first field is the timestamp when a license is checked out. The second field is the timestamp when the license is checked in again.

Field 4 and Field 5 together hold the total duration time of the activity. This first contains the time value and the latter holds the total number of days. When Field 4 exceeds 24 hours, it is reset, and the value of Field 5 is increased with one.

Field 6 is the name of the user.

Field 7 contains the setting of a project variable. The first part of the string contains the project name, followed by the name of the server where the license is taken from. This string also contains the date (month/year) of the log file and the job identifier from the executing task. The value of this variable is not always set, and therefore the value of this field is sometimes unknown (UNKN).

4.4.2 File organisation

Each license-checkout/checkin logs a lot of information, which is located on each license server. At the beginning of a new month, the binary data of the previous month is moved to a final compressed file, containing all transactions of the previous month. See Figure 4.8.
Setting up a data gathering infrastructure for PRODUKTIV+ 37

Figure 4.8 - License Log Data File Organisation

For each license server (site), log data is stored per month, per vendor. Unlike EDA tool log data, these files are well organised and the file locations are transparent. The likelihood to find duplicated data is as good as none. Also, these log files contain formatted text (comma separated). The lifetime of a compressed log file is at least 12 months.

4.5 Conclusion

This chapter described the data sources of the system. The first, EDA tool log data, is the most complicated data source. The variety of different log file formats is unlimited. Therefore, these log files are classified either as a type level-1, level-2 or level-3 log file. Level-1 log files turned out to contain the most relevant information, all together in a single file. Unfortunately, this file is only available (and complete) at the end of the design project run. A level-3 log file contains the highest amount of details. This file is created after completion of a single task. Level-2 log files are created when a tool execution is completed. This log file contains a summary of all the executed tasks (level-3 log files).

Important log information found in EDA tool log data are LSF results. When a task is submitted to a LSF queue, this action is written to the log file. After completion of the LSF job, a LSF summary is written the log file. The most important parameter in this summary is CPU usage. This indicates the duration of the task (in seconds).

The second data source is license log data. This data source is less surprising, and its format is more transparent. In the system, software licensing is managed by GLM (Global License Management). Each license transaction (checkin or checkout) is stored on the license server. License transactions are organized (and stored) per vendor. At the beginning of a new month, the server creates a compressed file containing all the transitions of the previous month (one file per vendor). This data source is very reliable,
ant its content is complete. The format of the log data is a collection of comma separated
files, whereby each field is well explained.

This chapter answers the important business question ‘where’ log files can be found, and
‘what’ its format is. The next chapter is focused on the architecture. In this chapter it
becomes clear how this log data is used.
5 Architecture

This chapter describes the data gathering architecture. One important part of this chapter is data mapping. So called ‘source schemas’ are mapped to ‘target schemas’. The main source is log data, described in the previous chapter. The target schema is the database management system (DMBS).

![ETL Concepts and Relationships](image)

**Figure 5.1 - ETL Concepts and Relationships**

Figure 5.1 illustrates the main concepts of this ETL architecture. ETL stands for Extract, Transform and Loading, and is explained later on. Blue circles represent a source schema, and the green circle represents a target schema. An arrow indicates an activity or action. The figure shows two different layers. The upper layer illustrates a source schema, undergoing various transitions \(S, S', S''\) to finally map to its final target schema. Note that the sequence of the chosen actions here is just an example. Also, it is not just one source schema and one target schema, in fact, both can be many. The lower part displays the layer which enables a tool to perform these transitions. In this figure, the ETL tool is responsible for moving data from \(S\) to \(T\). These concepts shape the basic outline of this chapter.

5.1 Source Schema

There are two source schemas to consider, EDA tool log data \((S_1)\) and license log data \((S_2)\). The first is somewhat misleading, because of the high variety of possible log files. To be formally correct, this schema contains \(1..n\) different schemas to model all possibilities. For example, a detailed log file could contain a parameter \(xyz\), while the same parameter is not included in the summary log file. Therefore, it would require two different schemas to model both scenarios. In this case, the schema is based on log files of the type level-1 (paragraph 4.3.1). This log file type is selected, because it already contains all the desired parameters.

Table 5.1 shows two source schemas. Source \(S_1\) contains two attributes, Section and SubSection, which provide context information. This concept of context information is explained in more detail in the next paragraph. Attribute Param1..n can be any list of selected parameters from the log file. ExitCode is one example of a useful parameter, because it contains information whether a task execution went successful or not. Source schema \(S_2\) is quite straightforward, and the schema is always the same.
The previous source schemas are mapped to a target schema. The logical model of the target schema is shown in Figure 5.2. In this model, the extracted tool log data (S1) needs the most additional explanation.
ToolLogFile

The concept ToolLogFile contains three important attributes. The first, ID, is the unique identifier of the physical file. One important reason for this unique identifier, is to prevent file and data duplication (e.g. when extracting the same data, from the same file for a second time). The attribute Project relates the extracted data to a single unique project. This attribute makes it possible to merge data from different log files, and relate it to the same project. The Type attribute describes the log file type (e.g. level-1).

Section, SubSection

As mentioned earlier, this model is based on level-1 log file types. It contains two important context blocks, Section and SubSection. The first typically addresses the current tool. The latter describes the current task of the tool. The attribute Name contains the section or subsection name, which relates to a tool or task respectively. Sequence is the attribute that holds the sequence of occurrence in the log file. It helps to prevent data duplication, when data is extracted from a file for the second time. The attribute LogFileID associates the section with exactly one log file.

Param1, Param2, Param_n

These three concepts model extracted parameters. The general idea is to store the name, and value an extracted parameter. The attribute Type contains the parameter data type (e.g. String, int). Currently, all extracted data is of the data type String. This attribute is included for future purposes. The attribute SubSectionID add meaning (or context) to the parameter.

---

**Figure 5.3 - Context Blocks**

Figure 5.3 illustrates the basic principle that a log file contains sections, a section contains subsections, and each subsection contains parameters. Let’s assume that the first parameter, Param1, contains the exitcode of a task (e.g. it indicates whether a task completed successfully or not). Extracting all ExitCode parameters from the log file is fairly easy, and would result in a list of ExitCode parameters. Nevertheless, a single
ExitCode is useless without having additional context information. Therefore, by relating the context to the parameter, it becomes meaningful.

LicenseLogFile

The concept LicenseLogFile contains the log data from the licensing log. Attribute ID is the unique identifier of the original source. Its main reason is to prevent data duplication. The attribute Project relates the log data to the project. One important ETL realisation, is that log data with no project information is already filtered out. The attribute LSFJob contains the ID of the executed LSF job. It is required to be able to relate the license information to a certain context (task). The attribute Feature contains the (unique) name of the license. This license feature is associated with exactly one tool. The attributes StartTime and EndTime contain the time when the license is checked out, and checked in.

5.3 Extract, Transform and Load (ETL)

The process of acquiring data and converting it into useful, compatible and accurate data is often labelled ETL (for Extraction, Transformation, and Load).

Extraction is the task of acquiring the data (in whatever format might be possible) from the source systems. This can be as simple as dumping a flat file from a database or spreadsheet, or as sophisticated as setting up relationships with external systems that then supervise the transportation of data to the target system.

Transformation is often more than just converting data formats (although this is a critical step in getting the data to your system). Data from external systems may contain incompatible or incorrect information, depending on the checks and balances that were in effect on the external system. Part of the transformation step is to cleanse or reject the data that does not conform. Common techniques used as part of this step include character examination (reject numeric values that contain characters) and range checking (reject values outside of an acceptable range). Rejected records are usually deposited in a separate file and are then processed by a more sophisticated tool or manually to correct the problems. The values are then rolled into the transformed set.

Load is the phase where the captured and transformed data is deposited into the new data store (e.g. warehouse, mart).

The ETL process can be as simple as transferring some data from one table to another on the same system. It can also be as complex as taking data from an entirely different system that is thousands of miles away and rearranging and reformatting it to fit a very different system. Describing these steps is an intensive and time consuming job, and several approached are already described in different papers [6][7].

Figure 5.4 shows the ETL architecture. The two clouds represent a data source, with S1 and S2 representing its schema. T1 is the target schema in the architecture. The DBMS block on the right side is the database management system. The two most important components of the DBMS are the database (P+) and data warehouse (DW). It also contains a staging area (i.e. temporary database tables), what is used to perform ETL steps within the DMBS. The format of the XML intermediate files is defined in a DTD
Setting up a data gathering infrastructure for PRODUKTIV+. The yellow arrow indicates that these XML files are also used for other purposes (outside the scope of the thesis).

ETL Step 1a: EDA tool log data to XML

The main task of ETL step 1a, is to extract data from EDA tool log files and store it in an intermediate file (XML document). Storage in an intermediate file is a constraint of the PRODUKTIV+ project, because other involved partners are going to use these files for other purposes (e.g. reporting). Another important constraint is, that all extracted values are of the data type String. This means that at in this ETL step no type casting operations occur.

Since these files are unformatted (i.e. plain text, ASCII), and the variety in format is unlimited, it needs an advanced search mechanism to find and extract desired parameters. Also, it needs a set of extraction rules to apply on different files (of the same file type, e.g. a tool summary file). The most important transformation step, is to transform the extracted data into XML format. The XML format is specified in its DTD.
ETL Step 1b: XML to Database

The task of this ETL step, is to load the data of the XML documents into the DMBS. This requires a mapping from the DTD (XML document) to the DMBS. Several techniques are studied on the topic of ‘storing XML in relational databases’ [9][10][11]. A common approach, is to store the XML documents itself in the relational database. Nowadays, various techniques are available to query or update these documents directly. However, these are not requirements of the system. While other partners intend to use these intermediate XML documents for other purposes, Infineon deletes these XML documents after the data is extracted and loaded into the database.

A simple XML reader can be used to retrieve the data from the XML documents. The process uses XSLT to transform the XML document into a table-based mapping as proposed in *Mapping DTDs to Databases* [11]. XSL stands for Extensible Stylesheet Language, and the T stands for Transformation [8]. This approach transforms the input into a generic target format. A great benefit for this approach, is that most DMBS’s (or commercials ETL tools) are able to handle this target format in an fully automatic manner, and therefore, processing these XML documents can be done in an automatic way. After this step, all data is located in the staging area (e.g. temporary database tables). Note that, at this point, all parameters are still of the data type String.

ETL Step 2: License log to Database

The main task of this step, is to extract log data from license log files, and load into the DBMS. This step is mainly focused on loading the data into the staging area. As described in paragraph 4.3.2, initially this log data is stored in compressed files. Since the programming language PERL is widely used over the company, it is suggested to use PERL scripts to decompress the archive. A second PERL-script is used to perform pre-processing, namely filtering out project-unrelated information. This action results in a set of comma separated files, containing license log data of selected projects. This filtering step is extremely important, because it reduces the size of log data enormous (i.e. Giga bytes to Mega bytes). After this ETL step, the pre-filtered license data is located in the staging area.

ETL Step 3: staging area (DBMS)

This step is responsible for moving data from the staging area to the database or data warehouse. Figure 23 suggests that, currently, only (automatically extracted) data is loaded into the data warehouse. The dotted arrow to the database indicates that the architecture is capable of it. Furthermore, the dotted arrow from the database to the data warehouse indicates that the data warehouse also retrieves data from the database. In this scenario, the database functions as a lookup table, e.g. to retrieve the unique project identifier, given a project alias.

The staging area contains one staging table for license log data, and multiple tables for tool log data. After the data is loaded into the data warehouse, all staging tables are deleted. This step is not described in more detail.
5.4 ETL Tool

An ETL tool is a software package that facilitates the performing of ETL tasks. Many commercial products and services are now available, and all of the principal database management system vendors now have offerings in these areas. Thus, a diverse range of tools exists with different capabilities such as SQL Server Integration Service, DTS, Data Stage, Oracle Data Warehouse Builder, etc. These tools differ by their performance, data source integration capability, use complexity, platforms, and more [12]. Most of the tools are found capable to perform the basic transformation and loading steps. Therefore, selection the appropriate tool depends on the already implemented technology. For instance, if the company already uses a Microsoft SQL Server database, then it benefits to select an ETL tool from the same vendor [13].

Nevertheless, these existing tools do have limits, for instance regarding maintenance. Dealing with updates is complex, because usually, a set of different programs have to be updated. Also, the EDA tool log data is too complex to handle with standard tools. Therefore, an ETL tool – the data extractor – is designed to perform additional specific processing in the data gathering architecture. As the name implies, it focuses particularly on the data extraction. The data extractor is capable of advanced finding and extracting of parameters from a selected data source. Extraction rules are defined, to describe a data flow. The data flow describes the process that extracted data undergoes before outputting it to its final target. The data flow works in a similar way as the pipes-and-filter pattern in software engineering [14].

The designed data extractor (ETL tool) currently provides four different building blocks. A building block, or shortly block, is a component of the tool that provides functionality. The output of component $n$, can be connected to the input of component $n+1$. All blocks connected together describe the data flow of the extraction process. All components have shared properties (e.g. Name and BlockType) and additional properties (i.e. block specific). Each component has an `Execute` method, which is able to trigger other components’ `Execute` method. The following four components can be used:

- Source component
- RegEx Block Iterator
- RegEx Parameter
- Target component

**Source component**

![Text File Source](image)

**Figure 5.5 - Source Component**

**Properties:**

`SourceFile` path and filename of the source file

The source component is usually the first component in the data flow. The property `SourceFile` is set to specify the input file. When this component is executed, it reads the data from a specified source, and outputs the content to the next component in the data flow.
RegEx Block Iterator

Figure 5.6 - RegEx Block Iterator Component

Parameters:
- **StartExpression**: regular expression of the block start
- **EndExpression**: regular expression of the block end

This component extracts text blocks from the input stream. The start and end of a block is specified in the properties **StartExpression** and **EndExpression**, by means of a regular expression [15]. Regular expressions are an extremely powerful tool for manipulating text and data. In this component, when the regular expression matches both conditions, start and end, it outputs the complete text block to the next component in the data flow. It acts iterative, meaning that it passes each matched block to the next component.

The main motivation for this component is for being able to extract a certain context from a source file. This mentioned before, context information adds important meaning to extracted parameters.

RegEx Parameter

Figure 5.7 - RegEx Parameter Component

Properties:
- **ParameterName**: name of the parameter
- **ParameterExpression**: regular expression to find the value

This component searches for a specified parameter in the input stream. A regular expression is specified in the property **ParameterExpression**. Whenever the expression is matched, it extracts and passes the value to the next component. The key motivation for this component is to be able to extract design parameters from log data.

Target component

Figure 5.8 - Target Component
Properties:

TargetTableName  name of the parameter

The target component is used to write the value of a parameter to the database. To be more precise, it loads the staging area of the DBMS with extracted data. It also allows block parameters to be written to the database, what is required to be able to relate all extracted parameters afterwards.

Component parameters

Each component can have additional parameters attached to it. In this context, a parameter is related to a certain component, and is used to pass additional information from one block to another. A parameter can have a static value, or can point to a different component by referencing to it. The latter enables the system to pass context information from one component to another. Whenever the RegEx Block Iterator component moves into the next iteration, it puts the current iteration value in an output value which can be used to relate extracted values. This is illustrated in Figure 5.9.

Figure 5.9 - Example of Parameter passing in Data Extractor

Static parameter values can be useful when certain data cannot be automatically extracted, for instance the unique project name. By adding a component parameter with the project name, the staging area can benefit from this additional information.

5.4.1 Relational model

Data extraction projects are stored in the database, whereby a set of rules define how data from a certain source have to be processed. Rules can be reused and maintained by different users. The idea is adopted from Query-based Data Warehousing Tool [12]. The relational data model is shown in Figure 5.10. Database terminology is used to explain the model.

PROJECT

This table contains the name of each project. One project describes exactly one data flow.

REGULAR_EXPRESSION

This table contains a list of predefined regular expressions. The column ID is the unique identifier of each expression. Expression contains the regular expression itself, and Description stores a short description of the expression. It is useful to distinct different versions.
**BLOCK_TYPE**

This table contains the valid block types (components) in the data flow. At the moment it contains four rows, one row for each type described earlier. This table functions as an enumeration table, whereby new component types can be added in the future.

**BLOCK**

This table stores information of each single block. The column **ID** is the unique identifier of each block. **ParentBlockID** refers to the previous block in the dataflow. One important constraint of this column, is that it does not allow ‘NULL’ values (unless the block type is of the type source_component). The model contains a *one-to-many* relationship from **ID** to **ParentBlockID**. In terms of inputs and outputs, it means that a block has exactly one input and can output to multiple following blocks. The column **Sequence** is initially meant to determine the execution order of blocks. While this is an ‘easy’ approach to do it, it becomes unnecessary when the execution order is determined by the way how component inputs (and outputs) are connected. Nevertheless, this column is still useful when two (or more) input sources are used, to determine the first block to start. Attribute **BlockType** refers to the type of the block in the **BLOCK_TYPE** table. Name is the column which holds the (display) name of the block. The columns **SourceFile**, **RegexBlockStartID**, **RegexBlockEndID**, **ParameterName**, **TargetTableName**, and **RegexParameterExpressionID** are used to store the additional block type specific properties.

---

**Figure 5.10 - Data Model of Data Extractor**
BLOCK_PARAMETER

The column \texttt{ID} is the unique identifier of each block parameter. \texttt{BlockID} associates the parameter to a block. This is a \textit{one-to-many} relationship, whereby each block can have multiple block parameters, and each parameter belongs to exactly one single block. Column \texttt{Name} contains the name of a block parameter, and \texttt{Value} contains its value. Column \texttt{ByReference} is of the data type \texttt{Boolean}, meaning it determines whether the parameter value is static, or referenced to a different block parameter from a different block. In case of the last scenario, the unique identifier of the referenced block parameter is stored in the column \texttt{RefBlockParameterID}.

Note that the model contains an \textit{inheritance structure}, whereby each block type inherits from a base block. Database inheritance scenarios are described in paragraph 3.1. In this model, the hierarchy is mapped to a single table. The main reason for this choice is the ease for implementation. The \texttt{BLOCK} table will certainly not contain millions of different rows (component instances), and therefore data redundancy is not an issue. Nevertheless, if many new components are added in the future, it might become confusing.

5.5 Conclusion

This chapter described the ETL architecture of the system. Four different ETL steps are defined, for the design to work, and this answers an important research question. The goal of these ETL steps is to move data from a source location to a target location. During this process (also referred to as \textit{ETL processing}), data is transformed in many different ways. The tool that is able to perform these typical operations, is called an ETL tool.

Paragraph 5.4 described the designed ETL tool. Although existing commercial ETL tools provide good functionality, it is insufficient to meet all requirements. Especially when it comes to data extraction, most of the existing tools lack the functionality to find (and extract) information in an advanced manner.

The designed ETL tool consists of four different components that together provide powerful functionality to extract data from any complex log file. The tool is able to define a data flow, and store it in the database as an \textit{ETL project}. This data flow contains at least one source component, multiple extraction components, and one or more target components to store extracted data in the database. It describes ‘where’ to extract the data, and ‘what’ to do with it.
6 Database

The design process of a database design involves many important phases, e.g. planning and analysis, conceptual-, logical- and physical design or implementation. This process is too large to cover all completely in the scope of this thesis, and therefore a key focus has been set. The key focus of this chapter is the relational data model, i.e. the logical design.

The data model is led by the PRODUKTIV+ ontology, as mentioned earlier in the project description. The ontology plays an important role in this chapter, because its class model is the main input for the design of the relational model. This chapter describes an adopted methodology to map this class model to a relational model.

6.1 Mapping from the Class Model to the Relational Model

A class model is a model based upon a class system. It describes the types of objects in a system and their relationships. In this context this is a synonym for an UML class diagram. UML class diagrams model class structure and contents using design elements such as classes, packages and objects.

The relational model is a branch of set theory that deals with logical relationships between sets. The relational model represents data in the form of two-dimensional tables. Each table represents some real-world person, place, thing, or event about which information is collected. A relational database is a collection of two-dimensional tables.

This section describes a methodology to map a class model to the relational model [17][18][19]. This methodology is adopted from Database Modelling in UML [16] and describes a number of steps to go from the class model to the relational model. It suggests to only include the steps which are relevant in the current situation, and therefore to use this list for guidance. The methodology consists of the following steps:

1. Model Classes
2. Identify persistent objects
3. Assume each persistent class maps to one relational table
4. Select an inheritance strategy
5. For each class add a unique object identifier
6. Map attributes to columns
7. Map associations to foreign keys
8. Map Aggregation and Composition
9. Define relationship roles
10. Model behaviour

Since in this project the ontology already provides the class model, not all of these steps need to be taken.

6.1.1 Rational Rose UML Data Modeling Profile

The used data modeling technique is Rational Rose’s UML profile for Data Modeling [20]. It is a proposed UML extension to support the modeling of relational databases in UML. It includes custom extensions for such things as tables, data base schema, table keys, triggers and constraints. While this is not a ratified extension, it still illustrates one
possible technique for modelling a relational database in the UML. The basic terms and notations are described below.

**Tables**

![Figure 6.1 - UML Data Profile - Table display](image1)

A table in the UML Data Profile is a class with the `Table` stereotype, displayed as in Figure 6.1 with a table icon in the top right corner.

**Columns**

![Figure 6.2 - UML Data Profile - Table and Columns display](image2)

Database columns are modelled as attributes of the `Table` class. For example, the Figure 6.2 shows some attributes associated with the `Project` table. In the example, the column `ID` has been defined as the primary key, as well as two other columns, `name` and `startingDate`. Note that the example above defines the column type in terms of the native DBMS data types.

**Behaviour**

So far, only the logical (static) structure of the table is defined; in addition the behaviour associated with columns, including indexes, keys, triggers, procedures etc. should be described. Behaviour is represented as stereotyped operations.

Figure 6.3 shows our table above with a primary key constraint and index, both defined as stereotyped operations.
Note that the PK flag on the column ID defines the logical primary key, while the stereotyped operation '<<PK>> idx_project00' defines the constraints and behaviour associated with the primary key implementation (i.e. the behaviour of the primary key).

Adding to the example, additional behaviour such as triggers, constraints and stored procedures can be defined, as in the example below:

The example illustrates the following possible behaviour:

- A primary key constraint (PK),
- A foreign key constraint (FK),
- An index constraint (Index),
- A trigger (Trigger),
- A uniqueness constraint (Unique),
- A stored procedure (Proc) – not formally part of the data profile, but an example of a possible modelling technique, and a
- Validity check (Check).

Using the notation provided above, it is possible to model complex data structures and behaviour at the DBMS level. In addition to this, the UML provides the notation to express relationships between logical entities.
Relationships

The UML data modeling profile defines a relationship as a dependency of any kind between two tables. It is represented as a stereotyped association and includes a set of primary and foreign keys.

The data profile goes on to require that a relationship always involves a parent and child, the parent defining a primary key and the child implementing a foreign key based on all or part of the parent primary key.

The relationship is termed identifying if the child foreign key includes all the elements of the parent primary key and non-identifying if only some elements of the primary key are included.

The relationship may include cardinality constraints and be modelled with the relevant PK - FK pair named as association roles. An example is shown in Figure 6.5. Table GenericActivity is the parent, and table Activity its child.

![Figure 6.5 - UML Data Profile - Table relationships](image)

The following sections describe the steps to go from the class model to the relational model.

6.1.2 Step 1: Model Classes

The first step is to create a class model. It is the goal to engineer the relational database schema from this model. The class model captures associations, inheritance and aggregation between concepts.

In this project, the class model was given. The PRODUKTIV+ class model is a UML model created during the first work package of PRODUKTIV+. It is derived from the PRODUKTIV+ ontology, and it models the domain of ‘chip design’. Four different parties spend several months to create this model and its concepts.

The PRODUKTIV+ class model is confidential, and cannot be included in the thesis. Therefore, a non-confidential version is created and shown in appendix 12.2. Only a small percentage of the concepts is included, and most of the attributes are hidden or renamed. Nevertheless, it covers sufficient concepts, relationships and complexity to explain this mapping methodology.
The class model consists of the following entities (concepts):

- Activity
- Actor
- CellLibrary
- ConsumableResource
- DALibrary
- DesignArtifact
- DesignTeam
- Provider
- GenericActivity
- GenericFlow
- IPLibrary
- License
- Tool
- NonConsumableResource
- ToolLicense
- Task
- ToolUsage
- Project
- LicenseType
- ILibrary

6.1.3 Step 2: Identify persistent objects

In this step, elements that require persistence have to be separated from those that do not. Persistent classes are classes in an application that implement the entities of the business problem. Tool and DALibrary are typical examples of a persistent object. An example of a non persistent object is Resource.

**Persistent classes:**

- GenericFlow
- Project
- DesignTeam
- Actor
- DesignArtifact
- DALibrary
- CellLibrary
- GenericActivity
- Activity
- Tool
- NonConsumableResource
- License
- ToolLicense
- Task
- LicenseType
- ILibrary
- ToolUsage
- Project

**Non persistent classes:**

- NonConsumableResource
- ConsumableResource
- Resource

6.1.4 Step 3: Assume each persistent class maps to one relational table

In this step, assume each persistent class maps to one relational table. Therefore, each persistent class from the previous step is mapped to one relational table. This assumption works in most cases (leaving the inheritance issue aside for the moment). Possible exceptions to this rule (i.e. many-to-many relationships between two classes) are corrected in the following steps. In the simplest model a class from the logical model maps to a relational table, either in whole or in part. The logical extension of this is that a single object (or instance of a class) maps to a single table row. This is shown in the table below.

<table>
<thead>
<tr>
<th>UML class</th>
<th>Relational table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Actor</td>
</tr>
<tr>
<td>Tool</td>
<td>Tool</td>
</tr>
<tr>
<td>Task</td>
<td>Task</td>
</tr>
<tr>
<td>Project</td>
<td>Project</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 6.1 - Mapping Persistent Classes to Relation Tables

Table 6.1 shows only four mappings, but of course it has to be done for each persistent class.
6.1.5 Step 4: Select an inheritance strategy

Inheritance is perhaps the most problematic relationship and logical construct from the object-oriented model that requires translation to the relational model. The relational space is essentially flat, every entity being complete in its self, while the object model is often quite deep with a well-developed class hierarchy.

The deep class model may have many layers of inherited attributes and behaviour, resulting in a final, fully featured object at run-time. The following three inheritance strategies are studied in paragraph 3.1:

- Scenario 1: Map hierarchy to a single table
- Scenario 2: Map each concrete class to its own table
- Scenario 3: Map each class to its own table

The class model shows the relationship ToolLicense inherits from License. In this case, scenario 1 applies the best. It is easy to implement, and table update and retrieval is fast. This becomes more difficult when more license subclasses would show up if the future.

The second scenario is a good solution for modeling: IPLibrary inherits from DALibrary and CellLibrary inherits from DALibrary. As a result, two relational tables would be created, IPLibrary and CellLibrary respectively. It needs to be said that also scenario 1 could be a good solution for this example.

Given these two examples, it shows that two different scenarios apply ‘the best’ in two different cases. Nevertheless, it is suggested to apply the third inheritance strategy for all inheritance structures in the class model. The first motivation for this is to be consistent. By applying different strategies, it does not become clear from the class model ‘what’ strategy is applied. The second motivation for this is maintainability. Especially at this phase of PRODUKTIV+, its class model is still undergoing radical changes. By knowing that only one inheritance strategy is applied, it makes it easier to maintain. Therefore, the only applied inheritance strategy in this step is scenario 3.

For example, in the class model, Tool inherits from NonConsumableResource, and NonConsumableResource inherits from Resource. After this step, it results in the creation of three logical tables.

6.1.6 Step 5: For each class add a unique object identifier

In this step, unique object identifiers are assigned. In the relational model, the identity in the relational model is usually implemented with a primary key. To be consistent, the name of a primary key in each table is always chosen to be ID. For example, the class Actor currently has one column with the name name. It contains the name of the actor. In this step, a new attribute is added with the name ID. The most important characteristic of the primary key, is that it is unique (in the scope of only that table). Only when a separate table is used to model a many-to-many relationship, this rule does not always apply. Step 8 describes how to deal with this special case.

6.1.7 Step 6: Map attributes to columns

In this step, simple data attributes of a class are mapped to columns in the relational table. An example of a simple type is the attribute version of the class Tool. Its attribute type
(String) is simple to map to the appropriate data model data type (varchar). In fact, all simple data attributes can easily be mapped, by reading the data type from the class model and choosing the appropriate target data type from Table 6.2.

<table>
<thead>
<tr>
<th>Analysis</th>
<th>ANSI SQL 92</th>
<th>DB2</th>
<th>Oracle</th>
<th>SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>VARCHAR(255)</td>
<td>VARCHAR(255)</td>
<td>VARCHAR(255)</td>
<td>VARCHAR(255)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td>INTEGER</td>
<td>NUMBER(10,0)</td>
<td>INT</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>FLOAT</td>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>TIMESTAMP</td>
<td>DATE</td>
<td>DATETIME</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td>NUMBER(1,0)</td>
<td>BIT</td>
</tr>
<tr>
<td>BYTE</td>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td>NUMBER(3,0)</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SINGLE</td>
<td>FLOAT</td>
<td>REAL</td>
<td>FLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td>LONG</td>
<td>INTEGER</td>
<td>BIGINT</td>
<td>NUMBER(20,0)</td>
<td>INT</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>DOUBLE</td>
<td>FLOAT</td>
<td>MONEY</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2 - Analysis Object to Data Model Data Type Mapping

This table is taken from the UML Profile for Data Modeling [20]. In this step, each attribute from the object model is mapped to the corresponding SQL Server data type. For instance, an attribute of the type Boolean in the class model is mapped to an attribute of the type bit in the data model. For complex attributes (i.e. attributes that are other objects) we use the approach explained below for handling associations and aggregation.

6.1.8 Step 7: Map associations to foreign keys

In this step, associations are mapped to foreign keys. More complex class attributes (i.e. those which represent other classes), are usually modelled as associations. An association is a structural relationship between objects. For instance, look at the concepts ToolUsage, Tool and ToolLicense. ToolUsage has a Tool, and a ToolLicense associated with it. For each association, a new column is created, with a foreign key to the other table’s primary key. This example is depicted in Figure 6.6. In this example, PP is the given name of the database schema.

![Figure 6.6 - Example of Association Mapping](image-url)
6.1.9 Step 8: Map Aggregation and Composition

In this step, aggregation and composition relationships are mapped. Aggregation and composition relationships are similar to the association relationship and map to tables related by primary-foreign key pairs. There are however, some points to bear in mind.

Ordinary aggregation (the weak form) models relationships such as an Actor is member of one or more DesignTeams. In this instance, more than one person could be member of the same design team, and if the Actor ceases to exist, the DesignTeam associated with them will still remain to exist. This example parallels the many-to-many relationship in relational terminology, and is usually implemented as a separate table containing a mapping of primary keys from one table to the primary keys of another. The separate table is also known as an intersection table. An example is shown in Figure 6.7.

![Figure 6.7 - Example of Many-To-Many Case](image)

A second example of the weak form of aggregation is where an entity has use or exclusive ownership of another. For example, a Task entity aggregates a set of Activity entities. This implies a Task may be associated with zero or more activities from an Activity table, but each Activity may only be associated with zero or one Task. If the Task ceases to exist, the activities become un-owned or are passed to another Task. In the relational world, this could be implemented as each Activity having an owner column which stored a Task identifier. This is known as a one-to-many relationship, and is illustrated in Figure 6.8.

A weak aggregation could be implemented using either an intersection table (for the many-to-many case) or with a foreign key in the aggregated class/table (one-to-many case). In the case of the many-to-many relationship, if the parent is deleted, the entries in the intersection table for that entity must be deleted also. In the case of the one-to-many relationship, if the parent is deleted, the foreign key entry (i.e. owner column) must be cleared. In the table design, this is automatically taken care of when enabling the ‘cascading delete’ option for the relationship.
The class model has no composition relationships (i.e. the strong form of aggregation). Composition implies that an entity is composed of parts, and those parts have a dependent relationship to the whole. In UML, a composition relationship is depicted by a solid square on one of the both sides of the association. In the case of composition, the use of a foreign key is mandatory, with the added constraint that on deletion of the parent the part must be deleted also. Logically there is also the implication with composition that the primary key of the part forms part of the primary key of the whole.

6.1.10 Step 9: Define relationship roles

In this step, relationship roles are defined. For each association type relationship, each end of the relationship may be further specified with role information. Typically, you will include the primary key constraint name and the foreign key constraint name. This logically defines the relationship between the two classes. Figure 6.9 illustrates this concept.

Relationships are based on the PK-FK pair. This example relates a GenericActivity to an Activity by the appropriate primary and foreign keys. The assumption is that an Activity may only be associated with one GenericActivity.

6.1.11 Step 10: Model behaviour

In this final step, a decision has to be made, whether to map some or all class behaviour to the functional capabilities provided by database vendors in the form of triggers, stored procedures, uniqueness and data constraints, and relational integrity.

Relational databases from different vendors typically include some form of programmable SQL based scripting language to implement data manipulation. The two common examples are triggers and stored procedures. From a purely object-oriented point of view, it is decided to avoid triggers and stored procedures and place all
behaviour in the classes. This localizes behaviour, provides for a cleaner design, simplifies maintenance and provides good portability between DBMS vendors.

However, the concept of class inheritance is the exception to the rule. In this case delete-, insert-, and update functionality is implemented in the form of a stored procedure. For instance the entity Tool inherits from NonConsumableResource, which inherits form Resource. The effort to insert a new tool is high, because it requires three separate insert statements (one on each table), whereby the (new generated) primary key has to be passed form the first table to the two others. These operations are outlined below:

- Insert record in table Resource,
- Get new primary key value,
- Insert record in table NonConsumableResource,  
  (By using primary key value)
- Insert record in table Tool.  
  (By using primary key value)

By implementing the stored procedure spInsertTool, this saves a lot effort when implementing the classes.

After completion of all steps, the data model is as shown in appendix 12.3 and 12.4. From this point, other design process steps have to be considered. Another important step is to create a physical model. This step is well defined in several papers [19][20].

### 6.2 Architecture

The architecture of the Database Management System (DBMS) is based on the ANSI-SPARC architecture [21]. This is an abstract design standard for DBMSs. ANSI-SPARC stands for American National Standards Institute - Standards Planning And Requirements Committee.

![Figure 6.10 - ANSI-SPARC Architecture](image-url)
The ANSI-SPARC model of a database identifies three distinct levels at which data items can be described: an internal level, a conceptual level, and an external level.

**Internal Level**

The internal level covers the physical representation of the database on the computer (and may be specified in some programming language). It describes how the data is stored in the database in terms of particular data structures and file organisations.

**Conceptual Level**

The conceptual level describes what data is stored in the database and the relationships among the data. It is a complete view of the data requirements of the organisation that is independent of any storage considerations.

The conceptual level represents:

- All entities, their attributes, and their relationships.
- The constraints on the data.
- Security and integrity information.

**External Level**

The external level represents the user’s view of the database. It consists of a number of different views of the database, potentially one for each user. It describes the part of the database that is relevant to a particular user.

![Diagram](Figure 6.11 - Differences between Levels in ANSI-SPARC Architecture)

The overall description of the database is called the database schema. There are three different types of schema corresponding to the three levels in the ANSI-SPARC architecture. The first, the external schemas, describe the different external views of the
data. There may be many external schemas for a given database. The second, the conceptual schema, describes all the data items and relationships between them. There is only one conceptual schema per database. The last schema is the internal schema. At the lowest level, it contains definitions of stored records, the methods of representation, the data fields, and indexes.

Figure 6.11 illustrates an example of differences between levels. On top, it shows one external view, `vwTools`. This view is mapped to the tables `Resource` and `Tool` (external/conceptual mapping). As a result, the external view contains joined information from these tables. Relating this conceptual schema to the internal schema is called conceptual/internal mapping. This enables the DBMS to find the actual record or combination of records in a physical storage that constitute a logical record in the conceptual schema. In this example the structure contains a pointer `next`, to point to the next record, thus, it forms a chain of linked records. Note that the order might be different from that of the conceptual record.

The great benefit for using this architecture is to provide data independency. Changes to the conceptual schema are possible without having to change existing external schemas or rewrite application programs (i.e. logical data independence). Also, changes to the internal schema are possible without having to change the conceptual or external schemas (i.e. physical data independence).

### 6.3 Conclusion

This chapter described a methodology how to go from the class model to the relational model. In this thesis, the class model is an UML representation of the PRODUKTIV+ ontology. By studying this ontology, one of the research questions is met. This chapter also answers the question what the data model look like.

Although the PRODUKTIV+ class model itself is out of scope, this methodology is feasible to deal with future ontology changes. Practically this means that, given an updated class model, all steps have to be followed and applied if applicable.

Following the method of this chapter, a database for the PRODUKTIV+ class model was created. The database contains 98 tables, 5 views and 4 stored procedures.

The next chapter describes the data warehouse design. One important distinction between the database and the data warehouse, is the data it contains. While the database focuses on operational data, the data warehouse focuses mostly on extracted project data.
7  Data Warehouse

The key motivation for the data warehouse, is to set up a data store to collect extracted design data. The data warehouse is responsible for storing project data, so that it can be used in the next work package to create project A-Box instances.

7.1  Introduction

A data warehouse is a collection of data gathered and organized so that it can easily be analyzed, extracted, synthesized, and otherwise be used for the purposes of further understanding the data. Data warehousing is the process of designing, building, and maintaining a data warehouse system.

The data warehouse architecture describes the overall system from various perspectives such as data, process, and infrastructure needed to communicate the structure, function and interrelationships of each component. The infrastructure or technology perspective details the various hardware and software products used to implement the distinct components of the overall system. The data perspective typically diagrams the source and target data structures and aids the user in understanding what data assets are available and how they are related. The process perspective is primarily concerned with communicating the process and flow of data from the originating source system through the process of loading the data warehouse, and often the process that client products use to access and extract data from the warehouse.

The primary components of the majority of data warehouses are shown in Figure 7.1 and described in more detail below.

![Figure 7.1 - Primary Components of Data Warehouse](source: Wikipedia)
Source Data Layer

The source data layer contains the data sources. Data sources refer to any electronic repository of information that contains data of interest for management use or analytics. This definition covers databases (e.g. IBM DB2, Microsoft SQL Server, Microsoft Access, etc.), spreadsheets (e.g. Microsoft Excel) and any other electronic store of data. Data needs to be passed from these systems to the data warehouse either on a transaction-by-transaction basis for real-time data warehouses or on a regular cycle (e.g. daily or weekly) for offline data warehouses.

Data Transformation Layer

The data transformation layer is the subsystem concerned with extraction of data from the data sources (source systems), transformation from the source format and structure into the target (data warehouse) format and structure, and loading into the data warehouse. This process is known as ETL, and plays an important role in this thesis.

Data Warehouse Layer

The data warehouse is usually (but does not have to be) a relational database. It must be organized to hold information in a structure that best supports not only query and reporting, but also advanced analysis techniques, like data mining. Most data warehouses hold information for at least 1 year and sometimes can reach half century, depending on the business data retention requirement. As a result these databases can become very large.

Reporting Layer

The data in the data warehouse must be available to the organisation’s staff if the data warehouse is to be useful. There are a very large number of software applications that perform this function, or reporting can be custom-developed. Examples of types of reporting tools include OLAP tools and data mining.

Metadata Layer

Metadata, or ‘data about data’, is used not only to inform operators and users of the data warehouse about its status and the information held within the data warehouse, but also as a means of integration of incoming data and a tool to update and refine the underlying data warehouse model.

Examples of data warehouse metadata include table and column names, their detailed descriptions, their connection to business meaningful names, the most recent data load date, the business meaning of a data item and the number of users that are logged in currently.

Operations Layer

Data warehouse operations comprises of the processes of loading, manipulating and extracting data from the data warehouse. Operations also cover user management, security, capacity management and related functions. In the current architecture, it also contains two logical data marts. A logical data mart is a filtered view of the main data warehouse but does not physically exist as a separate data copy.
7.2 Business Question

The project PRODUKTIV+ describes the case where it intents to measure tool productivity. While the productivty formula is still ongoing research, certain design parameters are already expected to influence ‘tool productivity’. One of the goals in this thesis, is to contribute to this intention, by extracting tool log data. To achieve this goal, the following business questions are asked:

1. For how long did tool X use license Y during project Z?
2. What is the total CPU usage of tool X, during project Z?

The first business question targets to tool license issues. The second business question focuses on CPU usage.

This thesis gives no assumptions about how to interpret the results. That is rather the task of the data analyst. It is the intention to provide the crucial information.

7.3 Data Source selection

In order to build the data warehouse, the appropriate data must be located. Typically, this will involve both the current OLTP (On-Line Transaction Processing) system where the day-to-day information about the business resides, and historical data for prior periods, which may be contained in some form of legacy system. Often these legacy systems are not relational databases, so much effort is required to extract the appropriate data, also from external sources.

The following three data sources are selected in this architecture:

- Source 1: EDA tool log data
- Source 2: License log data
- Source 3: PRODUKTIV+ database

Source 1 is a collection of tool log files. These log files contain large amounts of unformatted text. This data source is necessary, because it contains crucial information to be able to answer both business questions, like the CPU usage of a design activity. In this case, it relates the LSF log information (CPU usage) to a specific tool and project. All characteristics of this data source are described in paragraph 4.3.

License log data (source 2) contains information to relate license usage to a project. To some degree, license log data can be seen as a large collection of comma separated files. Unlike the first data source, license log data is well-formatted, and already suitable for automatic data processing. See paragraph 4.4 for more detailed information regarding license log files.

The last data source is the PRODUKTIV+ database. This is a relational database, designed in this project. The database functions as an operational database, and forms an important input for the data warehouse, because it represents the company’s current state. Most tables in this database are used by the data warehouse as lookup tables. In other words, this source is necessary to merge, filter, or complete data from (and with) the other sources. Example: during the project, project names might change. By storing
Setting up a data gathering infrastructure for PRODUKTIV+

project aliases in the database, historical data can be related to the correct project. The database design is described in chapter 6.

Table 7.1 shows what data sources are required to answer what business question.

<table>
<thead>
<tr>
<th>Business Question 1</th>
<th>Source 1</th>
<th>Source 2</th>
<th>Source 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1 - Business Questions’ data source dependencies

7.4 Modeling

There are several modeling strategies for data warehousing projects. Two different modeling strategies are studied during the literature study, namely ER-modeling and dimensional modeling. The latter is used to model the data marts, because of the flexibility and understandability of the star schema [22].

Figure 7.2 - Dimensionally modelled Data Warehouse

Figure 7.2 illustrates the dimensionally modelled data warehouse. The arrows imply that a certain process or activity occurs, starting on the left and moving to the right. The three data sources are just explained. ETL-processing (ETL) is responsible for extracting the data from the sources and loading it into the data warehouse. This process is described in chapter 5. After the ETL process, the data warehouse is refreshed and both data marts contain updated data.

Data pre-processing is part of ETL processing, but in the context of data warehousing there are two important aspects to notice:

- **Standardization**: different sources create date- or timestamps in various formats. These data and time values must be standardized.

- **Filtering**: incomplete data must be filtered out. For instance, license log data contain a column with the project name, which is optional. Without this project information, it cannot be related to its actual project and therefore have to be filtered out.
### 7.4.1 Star schema

Both data marts are described in a star schema. A star schema is the simplest data warehouse schema, consisting of a single *fact table* with a compound primary key, with one segment for each *dimension* and with additional columns of additive, numeric facts. The star schema makes multi-dimensional database (MDDB) functionality possible using a traditional relational database.

The star schema for Data Mart A is shown in Figure 7.3. It contains the fact table *Usage*, and four dimensional tables. The fact of this schema is ‘license usage duration’. The star schema for Data Mart B is shown in Figure 7.4. The fact in this schema is ‘CPU duration’. Both schemas are explained in more detail in the next paragraph.

![Figure 7.3 - Star Schema 1](image)

![Figure 7.4 - Star Schema 2](image)
7.4.2 Logical model

The next step in data warehouse design is to create logical data models. The logical model is derived from the conceptual model, in this case the star schema. The logical model for Data Mart A is shown in Figure 7.5.

![Data Mart A Diagram](image)

The model contains a fact table and four dimensional tables. Each dimension has a key (in red), specially designed for the dimension itself. As the figure illustrates, the value of the key (in the fact table) is not imported from the operational database. Instead, the value of the key is a unique system-generated semantic-free identifier. This feature insulates dimensions from possible changes in the way operational keys are defined (and possibly redefined) in operational databases over time. The system-generated key also has a role in tracking the history of changes in dimensions’ records.

**Time**

The **Time** dimension table contain the attributes Year, Month and MonthPart. This last attribute needs some more explanation. In the company’s current license model, license costs are based on a certain license usage over a period of time. Each month is divided into two month parts, identified with the attribute MonthPart. When the date is between the first of the month until the 15th of the month, the value is 1. The value is 2, when the date is between the 16th and the end of the month.

**Project**

The **Project** dimension is similar to the **Project** table of the database (source 3), with one difference that it only contains the two (necessary) attributes. ProjectID contains the original key value from the database. The Name attribute is self explaining.
**Tool**

The Tool dimension is similar to the Tool table of the PRODUKTIV+ database, or to be fully correct, it is similar to vwTools. Unnecessary attributes are absent. vwTools is a database view that joins data from the database Tool table, together with other tables. ToolID contains the original database key. The Name attribute is self-explaining.

**License**

The License dimension is similar to the database view vwLicenses. This database view joins information from the database License table, together with data from a few other tables. Only the mandatory attributes are derived. LicenseID contains the original database key. The Feature attribute contains the name of the license feature.

**Usage**

The fact table Usage consists of foreign keys to the dimension tables Time, Tool, License and Project. In addition it has an attribute Duration, which represents the duration of the license usage.

The logical model for Data Mart B is shown in Figure 7.6.

---

**Figure 7.6 - Data Mart B**

**Project, Tool**

These dimensions are the same as described for Data Mart A.

**Task**

The Task dimension contains the name of the tools’ executed task. Unlike most other Name attributes in the schema, this name is retrieved from the tool log file itself.
**CPU Usage**

The fact table **CPU Usage** consists of foreign keys to the dimension tables **Tool**, **Task** and **Project**. In addition it has an attribute **Duration**, which represents the duration of the CPU usage. Since the relevant time values are year/month/date, an attribute **Date** of the data type **DateTime** is added.

### 7.5 Data quality & considerations

These are the most important data quality considerations:

**Completeness:**
- missing information might result in an incorrect interpretation of one of the business questions (e.g. missing LSF job give a wrong indication of CPU usage of a tool).

**Reliability:**
- extracted values should always belong to the correct context (e.g. tool or task information)
- avoidance of data duplication (since different sources might contain the same information)

**Usability & Maintainability:**
- easy enough to maintain, and
- flexible enough to add sources or functionality in the future.

### 7.6 Example

#### 7.6.1 Table data

Below are five tables, used as an input for the data warehouse, with a small example dataset for each table. Since we are not dealing with just relational databases as an input source, some sources don’t produce a physical output table as show below. Therefore, it is probably better to think of these tables as part of the temporary tables in the staging area. This is an important realisation, because shown table content might already be derived from multiple sources.

**Tools**

<table>
<thead>
<tr>
<th>ToolID</th>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tool A</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>Tool B</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Tool licenses**

<table>
<thead>
<tr>
<th>LicenseID</th>
<th>ToolID</th>
<th>LicenseFeature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>FEAT A</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>FEAT B</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>FEAT C</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>FEAT D</td>
</tr>
</tbody>
</table>
### Project

<table>
<thead>
<tr>
<th>ProjectID</th>
<th>Name</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project X</td>
<td>01-08-2006</td>
</tr>
<tr>
<td>2</td>
<td>Project Y</td>
<td>01-12-2006</td>
</tr>
</tbody>
</table>

### Tool log

<table>
<thead>
<tr>
<th>Project</th>
<th>Activity</th>
<th>Tool</th>
<th>StartTime</th>
<th>EndTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project X</td>
<td>Activity 1</td>
<td>Tool A</td>
<td>31-08-2006 05:38:14</td>
<td>01-09-2006 14:10:06</td>
</tr>
<tr>
<td>Project X</td>
<td>Activity 2</td>
<td>Tool B</td>
<td>01-09-2006 14:10:07</td>
<td>01-09-2006 18:22:12</td>
</tr>
</tbody>
</table>

### Licenses log

<table>
<thead>
<tr>
<th>LicenseFeature</th>
<th>StartTime</th>
<th>EndTime</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEAT_A</td>
<td>31-08-2006 05:38:14</td>
<td>31-08-2006 16:38:01</td>
<td>Project X</td>
</tr>
<tr>
<td>FEAT_B</td>
<td>01-09-2006 11:14:00</td>
<td>01-09-2006 12:54:09</td>
<td>Project X</td>
</tr>
</tbody>
</table>

### 7.6.2 Data Warehouse

The tables below represent the physical tables of the data warehouse for Data Mart A. It shows an example of how the data could look like, based on the sample data from the previous section.

#### Tool

<table>
<thead>
<tr>
<th>ID</th>
<th>ToolID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Tool A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Tool B</td>
</tr>
</tbody>
</table>

#### License

<table>
<thead>
<tr>
<th>ID</th>
<th>LicenseID</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>FEAT_A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>FEAT_B</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>FEAT_C</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>FEAT_D</td>
</tr>
</tbody>
</table>

#### Project

<table>
<thead>
<tr>
<th>ID</th>
<th>ProjectID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Project X</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Project Y</td>
</tr>
</tbody>
</table>

#### Time

<table>
<thead>
<tr>
<th>ID</th>
<th>Month</th>
<th>MonthPart</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>2</td>
<td>2006</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1</td>
<td>2006</td>
</tr>
</tbody>
</table>
Using OLAP cube queries [23], one can discover for how long Tool B used license FEAT_B during Project X (see gray fields). In the example above, the query retrieves the license with the ID 2, tool with the ID 1, project with the ID 1, time with the ID 2, and the duration of 5001 seconds.

### 7.7 Conclusion

This chapter described the data warehouse design. Based on the log data, it succeeded to design a data warehouse for answering both business questions. These results are evaluated in chapter 9. This chapter also answers the research questions what data sources are available, and what data to store in the data warehouse. In the next chapter, two prototypes are developed. One of the prototypes is based on the data warehouse design, and is responsible for producing the data for answering both business questions.
8 Prototype

This chapter describes the prototype of this project. There are two prototypes developed, respectively PRODUKTIV+’s and IFX’s web application. Both prototypes have the same architecture, and use a similar technology. The first prototype is necessary to verify the database design. The second prototype is necessary to verify the data extraction process. For instance, it produces result data to answer both business questions. These prototypes are described in the next section. The architecture is adopted from the designed architecture in chapter 5.

The first prototype is about the database design, described in chapter 6. The relational model is implemented in a SQL Server database, and a maintenance application is developed in ASP.NET 2.0 technology. Users can manage typical ontology related information (e.g. tool or license information) via a web interface. This prototype is described in paragraph 8.1.

The following paragraph describes the second prototype. This prototype focuses mainly on data warehousing. The data warehouse is also implemented in a SQL Server database. A web interface (ASP.NET 2.0) provides functionality to set up extraction rules, extract information from a log file, and more. Simple reporting is implemented to provide a basic analysis interface.

8.1 Prototype I: PRODUKTIV+’s Web Application

The first goal of this prototype, is to implement the database design. The second goal is to provide a user interface to manage the data. There are several important software engineering aspects to consider when creating an application, e.g. error handling and user input validation. Although many aspects are considered, these are not in the scope of this thesis.

The database design for PRODUKTIV+ is described in chapter 6. This design is implemented in a SQL Server 2005 Express database, using the SQL Server Management Studio tool. The database includes all important concepts of the class model. The application is created in ASP.NET 2.0, with C# as the programming language. This web application provides functionality to manage the data in the database.

The database is designed in the context of PRODUKTIV+, and therefore all involved partners may apply to use the application. Nevertheless, partners cannot use the same database instance, because it contains company confidential information.

8.1.1 Graphical User Interface (GUI)

Users of the system access the application via a web browser. Styling and formatting these pages is certainly not the focus of this prototype. Nevertheless, both applications use a master page template, which encapsulates the page styling and lay-out, from where other pages derive. Because the master page includes a Cascading Style Sheet (CSS), it is fairly easy to (globally) change the layout at any time. Figure 8.1 shows a screenshot of the applications’ start page.

---

3 http://www.asp.net/
ASP.NET 2.0 provides a large amount of standard controls (web controls), that provide powerful functionality. The complete list of web controls can be found on the Microsoft MSDN website\(^4\).

Each page is specified in an XML file to create a site map. The `SiteMapDataSource` control is used to provide a data source interface to other controls. The top of the application contains a menu (`Menu` control). It allows quick access to different pages. The `Menu` control uses the `SiteMapDataSource` control to display its data. Just above the red dotted line is a navigation path (`SiteMapPath`). This navigation path is useful for the user to see where he or she is at. This `SiteMapPath` control also uses the `SiteMapDataSource` control to render its data.

The screenshot also shows the text ‘Overview’, this is the default position for the page title. The page content is always below its title. As the screenshot illustrates, the application is divided into eight parts. Each part represents exactly one part of the PRODUKTIV+ ontology. This way, users that are familiar with the ontology are able to find concepts in a rather intuitive manner.

---

\(^4\) [http://msdn2.microsoft.com/](http://msdn2.microsoft.com/)
8.1.2 Default navigation

The application uses a general way for navigating. From the start page, users can navigate to a **list page** (item list). From the **list page**, the user can navigate to a **detail page** (item detail), or go back to the home page. This is illustrated in Figure 8.2. Both pages are described in more detail, in the following section.

![Diagram of Default Page Navigation](image)

**Figure 8.2 - Default Page Navigation**

Users can perform actions to insert, update or delete items. Figure 8.3 shows the accessibility of different page actions on each page.

![Diagram of Page Action Accessibility](image)

**Figure 8.3 - Page Action Accessibility**

The action **Insert()** creates a new item. Action **Update()** is used to modify an item, and the action **Delete()** deletes an item. The action **DeleteItem()** deletes the selected item from the list. **InsertItem()** navigates to the item detail page in the Insert-mode. **UpdateItem()** navigates to the item detail page in Update-mode.

8.1.3 List page

As the name implies, a **list page** contains a list of items (e.g. tools, licenses, actors). Each list page has a **SqlDataSource** and a **GridView** control. The first control queries the database and holds the result in a dataset. The **GridView** control renders and visualizes the data in some tabular form. This is depicted in Figure 8.4. This control provides default functionality to enable sorting, or recordset paging of the displayed data.
Setting up a data gathering infrastructure for PRODUKTIV+

8.1.4 Detail page

A detail page (item detail) is a page that contains detailed information of an item. This is the page typically used to view, insert or update an item’s detail information.

8.2 Prototype II: IFX’s Web Application

This prototype covers the data extraction part of the project. Figure 5.4 shows the proposed architecture for data extraction. One important requirement for this architecture, is to have a flexible way to specify extraction rules. This is the main goal for this prototype.

The second goal for this prototype, is to implement a data warehouse with two data marts (as described in chapter 7). Both data marts are implemented to answer the two main business questions. Of course the architecture must be feasible to provide the data warehouse with extracted data.

The application provides three basic functionalities: quick analysis, custom extraction and data extraction projects. The application is also implemented using ASP.NET 2.0 technology together with a SQL Server database.
8.2.1 Graphical User Interface (GUI)

The same characteristics of the graphical user interface, described in the previous paragraph, are the same in this prototype. Figure 8.6 shows a screenshot of the start page of the application.

![Figure 8.6 - Screenshot of start page (Prototype II)](image)

As this figure illustrates, the application consists of the three main parts: ‘Data Sources’, ‘Data Extraction’ and ‘Reporting’. Data sources provide an interface to manage log data. Data Extraction is divided into three modules: Quick Analysis, Custom Expression, and ETL Project. These modules, or sub applications, are described in more detail in the following sections. Reporting displays the results of extracted data for answering both business questions. The functionality how the report itself is created, is not described in this chapter. However, the results of both reports are evaluated in chapter 9.

8.2.2 Log files

‘Log files’ is a page for managing log files. Each log file is stored in the database, so that multiple users can perform different analysis on the same log file. The page navigation is the same as shown in Figure 8.2. Users can insert new log files, edit log file meta info or delete log files. This page uses a FormView control to display its data. Figure 8.7 shows a screenshot of this web page.

![Figure 8.7 - Screenshot - Insert Log File](image)
A **FileUpload** control is used to provide upload functionality, to upload a selected file from the client to the server. This control renders the output as show in the picture above (after ‘File’). After selecting a file, the user selects the log file type (‘Type’). Log file types are described in paragraph 4.3.1. By specifying the log file type, the system is able to perform some default operations that only apply for the selected type. The fields ‘Project’ and ‘Description’ are currently optional, and give other users an idea what log file is used. After pressing the Insert-button, all information (including the log file itself) is stored in the database.

### 8.2.3 ETL project

This is the most powerful application of the three. This application allows a user to setup a complete ETL project (i.e. data extraction and *extraction rules*). The complete data flow can be specified by the user, by connection different components to form a pipeline, whereby each component manipulates the data in some way.

The first thing a user starts with, is specifying the data flow on a piece of paper or in a document. Typically, such a data flow starts with an input component, followed by several extraction components, and ending with one or more output components. For example, let’s assume that the user is interested in all *exit codes* of the executed project. Figure 8.8 shows a possible way to describe the data flow.

![Data Flow Diagram](image)

**Figure 8.8 - Example of Data Flow specification**

The first component, **LogFile**, is the input component (data source) of the data flow. This can be any log file, in this case a project report. The second component is of the type **RegexBlock**, which is a *block iterator* whereby the beginning and ending of a block are specified using a regular expressing. Regular expressions are available for most common programming languages, delivered in a library. The regular expression library for .NET is
used in this prototype\textsuperscript{5}. In this example, the component extract all sections from the input flow. The component below is also of the type RegexBlock. It extracts all subsections from the component input. Other components in the data flow are of the type RegexParameter. These components extract parameters, also by specifying a regular expression.

ETL project Setup Page

When the user is clear about ‘what’ data to extract and how the data flow looks like, he or she uses the ETL project setup page. Here the user can, step by step, build the complete data flow via the web interface. The basic structure of this page is outlined in Figure 8.9. ETL projects contain blocks. For each block, the user has to specify the block type, and other block attributes. Some attributes exist for all blocks, while other attributes are block type specific.

Each block can have zero or more block parameters. These parameters are used to pass additional values from one block to another. The user also specifies block parameter attributes, as the following screenshot illustrates.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure8.9.png}
\caption{Structure of ETL project Setup Page}
\end{figure}

\textsuperscript{5} http://www.regular-expressions.info/dotnet.html
When the complete data flow is specified, the user saves it and the project is stored in the database. Next, the user navigates to the 'Run project' page, where the project can be executed. The mechanism that executed the project is described in the next paragraph.

Two ETL projects are created to meet the two business questions of the thesis, one for each. The example data flow of Figure 8.8 does not include blocks of the type TargetOutput. TargetOutput blocks can be used to output parameter values to a database, whereby each block creates a temporary output table. The collection of temporary tables can be seen as the staging area of the architecture.

### 8.2.4 Run ETL project

The mechanism to execute ETL projects is written in the programming language C#, and contains the following important parts.

- **LogFileParser class**
- **ETLBlock class**
- **ETL project interpreter**

The LogFileParser class contains functionality to parse text files (log data) and extract information. This class is also used in the other two data extraction applications.

The ETLBlock class describes the different block types, and its properties. The base class implements the method Execute, to execute the block. It models the structure as outlined in Figure 8.9. The class is constructed in such a way that new block types can be added in the future, easily.

The ETL project interpreter retrieves all ETL project information from the database. First it retrieves all blocks, ordered by the block sequence. Second, it creates an ETLBlock instance for each block. Third, for each block it creates a list of ETLBlockParameter instances. And finally, it executes the first block. This procedure is described below.
It is not a goal in this prototype to write the ‘best’ interpreter possible. The main purpose of the interpreter, is to enable the proof of concept: setup a flexible way to extract data from log files.

After running an ETL project, the staging area is loaded with (temporary) data. From here, some additional steps have to be done to load the data into the data marts. This is not automated in the prototype. The following steps are important to perform this action:

Step 1: create views

Multiple database views are build on top of the staging tables. This is necessary to join (i.e. relate) the data.

Step 2: load/refresh data warehouse

Create SQL Insert statements to load the data into the data warehouse. Data is retrieved from the dedicated views (staging area) and load into the data warehouse.

After these steps, the staging tables are deleted again.

8.2.5 Quick Analysis

This application is developed to give a user an easy way to (quick) analyse log data. During the design project, tools produce a huge amount of log data. Usually, these text files are large (up to gigabytes) and therefore make it hard for the designer to open and easily find desired parameters. For example, let’s assume that a designer runs an InWay design, and the execution somewhere fails. At this point, the designer might be interested in knowing ‘what’ commands where executed and ‘when’. Rather than opening the log file in an editor, the designer can use Quick Analysis to analyse the file, and have a quick result.

- load all blocks
- for each block
  - create block instance
  - set block type
  - set block default attributes
  - set block type specific attributes
  - load block parameters
  - for each block parameter
    - set default attributes
    - set type specific attributes
  - detect whether startup block or not
- execute startup block
Figure 8.11 - Screenshot of Quick Analysis application

Figure 8.11 shows a screenshot of the application. First, the user selects a log file from a list. Secondly, the user specifies the desired parameters to extract (see Table 8.1).

Parameter selection:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time stamps</td>
<td>Start- and end time of task executions</td>
</tr>
<tr>
<td>InWay results</td>
<td>Exit code and duration of a task</td>
</tr>
<tr>
<td>Submitted Jobs</td>
<td>JobId of jobs submitted to a LSF queue</td>
</tr>
<tr>
<td>LSF results</td>
<td>Result of LFS job(s)</td>
</tr>
<tr>
<td>CMD</td>
<td>Executed command (and arguments)</td>
</tr>
<tr>
<td>LOG</td>
<td>Reference to (more detailed) log files</td>
</tr>
</tbody>
</table>

Table 8.1 - Available parameters in Quick Analysis

The next step for the user, is to select whether to extract context information or not. Paragraph 4.3.1 described abstraction levels of tool log files. When data is extracted from high level log files, it contains merged log data from multiple tool outputs. In this scenario the context is important, and therefore the user should check the ‘Include context’ box. When a detailed log file belongs to a single tool output, it contains only this context, and therefore the ‘Include context’ box can be left unchecked (to optimize for speed performance). After running the Quick Analysis, the screen might show the following result:

Figure 8.12 - Sample result of Quick Analysis
In this example, each extracted parameter is recognized by the ‘@’ sign. So therefore ‘ExitCode’ is a parameter, and ‘0 (0)’ is its value. And in a similar way, ‘Elapsed’ is an extracted parameter, and ‘1m 30s’ is its value. The parameters belong to a certain context, which can be found by reading upwards in this tree, until you find the ‘-’ sign. This represents the first context block, i.e. the name of the executed task. So in this example, the previously mentioned parameters belong to the task ‘power’. On its turn, the ‘power’ context belongs to a parent context which is the name of the tool. By reading upwards in the tree again, the top-level can be found, in this example with the name ‘Blastfusion’. In terms of grouping (database terminology), these results are grouped by: Tool, Task, Parameter.

### 8.2.6 Custom Expressions

This application follows the same principles as Quick Analysis, with one big difference that parameters are not fixed. See Figure 8.13. After selecting the input log file, the user can specify a custom regular expression to extract a desired parameter. This is useful in two cases. The first case, is when the user wants to extract a parameter that is not already pre-defined. The second case, is when the user uses this application to create or test a regular expression.

![Log file parser (custom)](image)

**Figure 8.13 - Screenshot Custom Expression**

In the example above, the user is interested in all command parameters (CMD: xxx) from the log file. The user specifies the regular expression [15] and presses ‘Run’. The results are shown in the result-box, in a similar way as for Quick Analysis.

### 8.3 Architecture

Both prototypes use a client-server architecture. The application runs on a web server, where multiple web clients access the application via a browser. The infrastructure is shown in Figure 8.14.

As the illustration shows, all web clients communicate with ASP.NET applications through IIS (for Internet Information Service, i.e. web server). IIS deciphers and optionally authenticates the request. IIS also finds the requested resource (such as an ASP.NET application), and, if the client is authorized, returns the appropriate resource.

ASP.NET is a web application framework marketed by Microsoft. Programmers can use it to build dynamic web sites, web applications and XML web services. It is part of Microsoft's .NET platform and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime, meaning programmers can write ASP.NET code using any Microsoft .NET language 6.

---

6 [http://support.microsoft.com/ph/8291](http://support.microsoft.com/ph/8291)
The ASP.NET application accesses a back-end SQL Server database. The application is used by corporate employees who have accounts within Active Directory. The Web application uses a trusted subsystem model and executes calls to the database on behalf of the original callers. The communication between the browser and the Web server is over HTTP. The application functions as an Intranet application, and therefore users are unable to access it from outside (WWW). See the architecture in Figure 8.15.

Figure 8.14 - Prototype Infrastructure

Figure 8.15 - Prototype Architecture
Configuration
The ASP.NET web application uses a configuration file, called ‘Web.config’, to keep the configurations required for the corresponding application. The ‘Web.config’ file is written in XML with specific tags having specific meanings. It stores settings like database connections, Session States, Error Handling and Security.

Web Server
A dedicated application pool (IIS) is used and configured to run under a custom domain service account with access to the database. The application’s virtual directory is configured in IIS for Windows authentication. Anonymous access is disabled. ASP.NET is configured for Windows authentication `<authentication mode="Windows"/>`. The database connection string is held in the `<connectionString>` section of the application’s ‘Web.config’.

Database Server
SQL Server configured for Windows authentication, which determines the identity of a user attempting to connect, based on a verifiable identifier. Authorization establishes the level of privileges granted to a login associated with the logged on user. A SQL Server login is created for the application’s application pool identity. The login is mapped to a database user for the Web application. The database user is placed in a database role for the Web application. Database permissions are granted to the database role.

Products overview
The following products are used:
- Microsoft ASP.NET 2.0
- Microsoft .NET Framework 2.0
- Microsoft SQL Server 2005 Express
- Windows NT/2000
- Microsoft IIS 5.1

8.4 Summary
This chapter describes two different prototypes. Both prototypes use Microsoft .NET technology. The first prototype is developed to implement and verify the database design. The second prototype provides functionality to extract data from log files, with the main goal to verify the reliability of the data extraction architecture. It is also responsible for creating the result data to answer both business questions. All these results are evaluated in the next chapter.
9 Evaluation
This chapter contains the evaluation of the results.

9.1 ETL architecture
This paragraph evaluates the designed ETL architecture. The two most important criteria to reflect on, are reliability and performance.

9.1.1 Reliability
Infineon’s system *InWay* is able to produce a test report, at the end of the design project run. This report contains a summary of all the executed steps in the design project. It consists of a list of tool drivers that are loaded during execution. For each tool driver, the start- and end time is shown, as well as the total duration (in seconds). This test report is reproduced in the prototype. Data is extracted from a summary log file, and the result is shown in appendix 12.5. The goal of this result is to verify the extracted results, and to proof reliability.

See appendix 12.5. The first column, *SectionId*, is the unique identifier of each *section* in the database. Column *Driver* contains the name of the tool driver, which always relates to a single tool. The column *Start* shows the time when the driver is loaded. Column *Stop* displays the time when the driver is unloaded. Column *Duration* contains the sum total of the step (in seconds). Gray-colored rows indicate that the row is not extracted automatically (i.e. it is added afterwards). These rows exist in the original test report, but are absent (i.e. not found) in the test result. Red-colored cells indicate an error. Yellow-colored cells indicate a remark.

Two sections (grayed background color) are missing in the result. The first, ‘Project Setup’, is the step of the project run where the complete project is configured in *InWay*. This step does not write any start or end log information of the section to the log file, and therefore it is cannot be found by the data extractor. The second missing section is ‘DC’. The reason why this section was not found, is because it is *skipped* during the project run. When the designer configures a project in *InWay*, steps of the project can be set to *skipped*, e.g. when an output is already present. However, this information is written to the log file, and therefore it should be possible to automatically detect its existence and include this in the test report.

The test result contains one error. Driver ‘ConfigProject’ contains a duration of 3 seconds. In the original report, the duration for this section is 0. Instead, the value of 3 is related to the previous step, ‘ProjectSetup’ (whereby this context information is not written to the log file). From inspecting the log file, the extracted value shown in the result is correct, and therefore this ‘error’ is explainable.

The test result contains two yellow-colored cells, which needs some explanation. The first, ‘ModelsimTDrv’, is related to a *section* with the *ID*=7. Above this line, there is already a different driver related to the same *SectionID* (*ID*=7). What it shows, is the occurrence of a nested driver (see Figure 4.4). Nested drivers are not shown in the original report, because the duration of the nested driver does not increase the total duration of the *section*. For example, consider the duration of the main driver to be 100
seconds. Now, consider the duration of the nested driver to be 20 seconds, somewhere in between the time span of the main driver. As a result, the duration of the nested driver does not contribute to the sum total for this section, because it still remains 100 seconds. The same thing happens at the second yellow-colored cell, showing the driver ‘ModelsimTDrv’ (SectionID=9).

9.1.2 Performance

The performance of the system is tested, by measuring the total data extraction time, based on different settings. The first setting to vary is the input file. Three different source files are used in this test case. The second setting to vary is the number of parameters to extract. The range of this setting is between 1 and 4 parameters. The last setting to vary, is the option whether or not to include the context information. If this value is set to true, the data extractor attempts to extract context information. If the value is set to false, no context information is extracted.

The following input files are used:

<table>
<thead>
<tr>
<th>File#</th>
<th>File name</th>
<th>Description</th>
<th>FileSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bind.log</td>
<td>Blastfusion detail file</td>
<td>408 kb</td>
</tr>
<tr>
<td>2</td>
<td>0Protocol.log</td>
<td>Blastfusion summary file</td>
<td>21 kb</td>
</tr>
<tr>
<td>3</td>
<td>Run_036.log</td>
<td>Run summary file</td>
<td>11830 kb</td>
</tr>
</tbody>
</table>

The following parameters are used:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param1</td>
<td>Execution start</td>
</tr>
<tr>
<td>Param2</td>
<td>Execution stop</td>
</tr>
<tr>
<td>Param3</td>
<td>Exit code</td>
</tr>
<tr>
<td>Param4</td>
<td>Submitted LSF job</td>
</tr>
</tbody>
</table>

The following table shows the number of occurrences in each file:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>File# 1</th>
<th>File# 2</th>
<th>File# 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param(1)</td>
<td>1</td>
<td>0</td>
<td>95</td>
</tr>
<tr>
<td>Param(2)</td>
<td>1</td>
<td>0</td>
<td>95</td>
</tr>
<tr>
<td>Param(3)</td>
<td>1</td>
<td>1</td>
<td>252</td>
</tr>
<tr>
<td>Param(4)</td>
<td>1</td>
<td>0</td>
<td>112</td>
</tr>
</tbody>
</table>

Given these settings, this test is executed. The procedure is described below.

- for i=1 to 3
  - select input file i
  - for n=1 to 4
    - extract Param(n) from input file i (with context)
    - extract Param(n) from input file i (without context)

This procedure is repeated ten times, and the average values are shown in Table 9.1.
Table 9.1 - Test results: Data Extractor execution times (in seconds)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Input File# 1</th>
<th>Input File# 2</th>
<th>Input File# 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Param(1)</td>
<td>2.04</td>
<td>0.46</td>
<td>0.13</td>
</tr>
<tr>
<td>Param(1,2)</td>
<td>2.52</td>
<td>0.94</td>
<td>0.16</td>
</tr>
<tr>
<td>Param(1,2,3)</td>
<td>2.95</td>
<td>1.42</td>
<td>0.22</td>
</tr>
<tr>
<td>Param(1,2,3,4)</td>
<td>3.04</td>
<td>1.48</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Column Parameters shows which parameters where used during the extraction. A plus-sign indicates that the context information is also extracted. A minus-sign indicates that the parameters where extracted without context information. The table shows the total execution time in seconds. From this results it becomes clear that the extraction time is mostly influenced by the fact whether to include the context or not.

These results give a good indication of the performance of the data extractor. However, these results are base on single-user usage. When multiple users use the system at the same time, the performance might decrease dramatically. The main reason for this, is because the system uses a single thread to handle multiple requests. For example, consider two different users, using the system at the same thing. User A is running an extraction, whereby a large input file is used and multiple parameters are selected for extraction. This will require a lot CPU power, and also internal memory. Therefore, when User B starts the extraction process, the extractor has to wait until the previous process is finished. In this scenario, the waiting time depends mostly on the CPU power of the machine (where the application is running).

Another aspect which influences the performance a lot, are the regular expressions that are performed. It should not be surprising that many different expressions can match the same desired value or result. For example, consider the text ‘200’. The following expressions can be defined to match this value:

- \d\d\d\d
- \d{4}
- 20\{3\}

In here, \d stands for digit (0-9). The first expression tries to match any digit for four times. The second expression attempts to four-times match a digit. The last expression first tries to match the character ‘0’, and follow by matching the character ‘0’ for four times. All of the three examples are correct, and it basically depends on the creativity (and knowledge) of the designer, what solution is chosen. However, these examples are rather easy. More complicated expressions might require e.g. a look-ahead. This can be useful when the desired values should end with a certain value, which should not be included in the result. For example, in the text string ‘.. 50 seconds’, someone could be interesting in, only, the value ‘50’. In this case, a look-ahead is used to find the text, but its value is not included in the result.
9.2 Business questions

9.2.1 Business question 1

Appendix 12.6 shows the result of the first business question. The report contains the columns: \( Y, M, MP, \) Project, Tool, License feature, and Duration, which are explained below.

- **Y**: Year of the activity
- **M**: Month of the activity
- **MP**: MontPart (see paragraph 7.4.2) of the activity
- **Project**: Name of the project
- **Tool**: Name of the tool
- **License feature**: Name of the license feature
- **Duration**: Total duration (in seconds)

This report is grouped by the columns \( Y, M, MP, \) Project, Tool, License feature (in this sequence). This means that results are displayed per year, per month, per \textit{monthpart}, per project, per tool, per license feature. Furthermore, the result is sorted by Duration (descending). This means that the highest duration value is always on top (per grouping construction). With Duration is meant the total time span of a checkin/checkout activity of a license feature. The report contains a subset of log data from the license sever. Note that the license server creates log data, and organizes its files per vendor. In this report, log data is merged from five different vendors.

The results in this report are as expected. However, there are a few points that need further explanation. Cells with a yellow-colored background indicate that the information is incomplete. The reason why the name of the tool is ‘unknown’, is because crucial information is missing, to be able to relate the data. For example, the duration of the license feature BLAST VIEW, for the project UNKNOWN, for the project projectX is 10627 seconds. But this is incomplete, because the value UNKNOWN is grouped. In fact, this value might consist of 2 or more tools, whereby each tool contributes to the sum total of 10627 seconds. This crucial information that seems to be missing, is EDA tool log data. It contains the LSF job identifier, which is mandatory to relate the license usage to the tool (and its \textit{section}). In the ‘live’ architecture, incomplete (or incorrect) values are filtered out. These results are included in this report, as an example.

9.2.2 Business question 2

Appendix 12.7 shows the result of the second business question. The report contains the columns: \( Y, M, D, \) Project, Tool, Task, and Duration, which are explained below.

- **Y**: Year of the activity
- **M**: Month of the activity
- **D**: Day of the activity
- **Project**: Name of the project
- **Tool**: Name of the tool
- **Task**: Name of the (executed) task
- **Duration**: Total CPU duration (in seconds)
This report is grouped by the columns Y, M, D, Project, Tool, Task (in this sequence). Results are displayed per year, per month, per day, per project, per tool, per task. Furthermore, the result is sorted by Duration (descending). This means that the highest CPU duration is always on top (per grouping construction). The input source for this report is ‘run_036.log’. This is a level-1 (summary) log file for the project T-HIT (in the result referred to as produktv).

This result includes all the steps (tasks) that are executed on a LSF server, during the design project run (in InWay). In this scenario it is insufficient to simply extract all LSF results from the log file. This is explained by an example, see the sample log file in the figure below.

Figure 9.1 - Primary Components of Data Warehouse

On top of the log file it shows the text “Starting Tool 'ToolA'”. This indicates that a new tool-context started, belonging to ToolA. Second, the text “### 'task1' started” starts the context of a new task, in this example task1. The text ‘Job <7103> is submitted to queue <batch>.’ shows that a LSF job (with the ID 7103) is send to the LSF server.

Let’s assume the user wants to know the total CPU usage of each task. By simply extraction the LSF results from the log file above, both LSF results are relate to the task task3, which is incorrect. This LSF summary belongs to task3, but the CPU usage of each single result belongs to two different tasks. To extract this LSF information the correct way, the following has to be performed:

1. Extract all submitted to JOB parameters, including its context
2. Extract all LSF results
3. Relate the data, by placing a database join on the column (parameter) JOB. This is illustrated in Figure 9.2.
The table `SubmittedJob` contains the column `TaskID`, which provides the context information. The column `JOB` is joined with the column `JOB` in the `JOB` table. This way, the LSF results contain the correct context.
10 Conclusions
This chapter will list the results which have been gathered during the various phases of
the research. At first, the initial research questions are recalled and answered. Secondly,
additional conclusions are drawn. Finally, this chapter ends with a recommendations
section.

10.1 Conclusions
The research questions from paragraph 1.5 are repeated, and answered below:

Architecture design:

- **Where can we find data sources?**
  There are two different data sources: EDA tool log data and license log data. The
  first can be found within the InWay architecture. Log data is written to various
  network drives, depending on the project settings. License log data can be found
  on the license server. Unlike EDA tool log data, these files are well organized.

- **What is the structure of EDA tool log files?**
  Three different abstraction levels are defined, to classify an EDA tool log file. The
  log file can be classified as a level-1, level-2 or level-3 log file. Each level
  contains a different amount of information. Level-3 log files contain the highest
  amount of detail information (of a single task). The scope of a single file is the
  task. Level-2 log files contain summarized information of multiple tasks. Each
  task in the file belongs to a single tool. Therefore, the scope of this type is a
  single tool. Level-3 log files contain log information from multiple tools. Also,
  the amount of detail tool information is higher, compared to level-2 log files.

- **What is the lifetime of a data source?**
  The lifetime of a data source depends on the type of data source. Log data from
  the licensing server is available for at least one year. The lifetime of EDA tool
  log data, depends on its type (i.e. level-1, level-2 or level-3). Level-3 log files
  have the shortest lifetime. These level-3 log files contain detailed task
  information, which are overwritten when the same task is repeated for the second
  time. Level-2 and level-3 log files append information, when tasks are repeated.
  Basically, it is up to the designer to decide when these log files are deleted.

- **What are the ETL processing steps?**
  The ETL processing steps for license log data is the following. Log data is
  extracted from the server, and decompressed into a temporary folder. This results
  in a set of comma separated files. Perl scripts are used to filter out relevant data
  (based on selected design projects to evaluate) and loaded into the database
  staging area. The ETL processing steps for EDA tool data is less transparent. At
  first, data is extracted from EDA tool log data and stored in XML files. Secondly,
  XSLT is used to extract the data from the XML files and transform it into a table-
  based-mapping (also in XML format). After this, an existing ETL tool can be
  used to load the (XML) data into the staging are of the database. At this point, the
  staging are contains data from both data sources. It is up to the user to decide
  how to continue. For instance, if the user is interested in tool license usage
  information (business question 1) then the data from both sources has to be
  related. This can be done in an automatic manner using an existing ETL tool, or
  by writing custom SQL-scripts.
Database design:

- **What does the ontology look like?**
  The PRODUKTIV+ ontology models the area of ‘chip design’. It exists of eight parts, also referred to as ontologies. For each ontology, an UML class model was derived. This class model formed an important input for this thesis.

- **What does the data model look like?**
  The data model looks almost the same as the class model. A methodology is described, how to go from the class model to the data model. Following the methodology, a database for the PRODUKTIV+ class model was created. The database contains 98 tables, 5 views and 4 stored procedures.

- **What information do we need to store?**
  In the next work package of PRODUKTIV+, A-Box instances are created to hold ontology related data. Therefore, the data model is provided with all the (relevant) concepts from the ontology, so that everything can be stored. At least all tools and licenses should be stored in the database. This is important for the extraction process, to retrieve the unique identifier of a tool or license, when extraction from a log file. For instance, the tool Blastfusion is sometimes referred to as ‘bfusion’, and sometime as ‘Blastfusion’. In both cases the same tool is meant. Providing the database with this information will definitely improve the quality of data extraction.

Data warehouse design:

- **What data sources are available?**
  These are the same as mentioned earlier: EDA tool log data, and license log data.

- **What does the data warehouse look like?**
  The data warehouse is dimensionally modelled using star schemas. The data warehouse contains two fact tables (license usage and CPU usage) and seven dimensional tables (Time, Project, Tool, Task, License).

- **What data do we store in the data warehouse?**
  The data to store in the data warehouse, is determined by the star schema of the data warehouse design. To be able to answer both business questions, the following data has to be stored: tool, (tool) task, license feature, project, CPU usage (duration), license usage (duration), and timestamp of each activity.

The results, evaluated in the previous chapter, proof that the data extraction architecture is reliable. A large amount of data is extracted from a summary log file, and is verified with an existing report file. Only three minor differences occurred, which were explainable. The first difference was cause by skipped sections in the design project. As the name implies, these sections are not executed during the run of the project, and does not write the default context log information to the log file, and was therefore not found by the data extractor. To correct this, the regular expression to grab this context have to be extended. The second difference was an incorrect duration value, displayed in the result report. After inspecting the original log file, it turned out that the extracted value was correct after all. Because in the previous step context information was missing, the duration value was added to the ‘incorrect’ context. The last difference was caused by additional tool driver information found in the log file. Nested tool driver have no influence on the total duration time of the project run. Nevertheless, the data extractor
simply extracted all driver log information from the log file, and therefore the report contained these nested driver results. To correct this, the data extractor has to be extended, so that it only stores driver log information from the top-level context. This is fairly easy to implement, because the scope of each result is already determined during the extraction process.

Most of the results to verify the data extraction architecture are produced, using the second prototype. It also provided the results to successfully answer both business questions, as described in the previous chapter. The database design is implemented in the first prototype, as a proof of concept. Using the suggested methodology, it succeeded to create and implement the PRODUKTIV+ database.

10.2 Recommendations

Two recommendations are given below.

**Improvement of the ETL architecture**

During the project, a lot of effort is put into the ETL step to extract data from EDA tool log files, and store the data in intermediate XML documents. And the next step, to load the data from the XML files into the database (see Figure 5.4, ETL step 1a-1b). However, to improve the extraction process, the designed ETL tool should be used to extract the data from the log files, and load extracted data directly into the database. This way, the intermediate XML-documents are no longer necessary. Whenever this XML document is desired, it can be generate directly from database. This is already successfully tested in this project.

**Improvement of the designed ETL tool**

The source component of the designed ETL tool should be improved, to increase the performance. Currently, the source component reads the content of the input file all at once. As a result, the system will probably collapse if the file size of the source file is larger then the size of the internal memory. Therefore, the system needs extended functionality in the component, which is capable of pre-scanning the input file, and passing its content, multiple times in multiple smaller parts. Using a simple buffer file-read approach, would avoid the system from crashing, but is not sufficient to increase its performance. Functionality is needed to pre-scan a source file, already searching for context block, and passing its content (in smaller parts) to the next component.
11 References


[22] Erick D. Slazinski. *Teaching Data Warehousing to Undergraduates – Tales from the Warehouse Floor*. Purdue University.

12 Appendices

12.1 Comparing database inheritance strategies

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Simple approach.</td>
<td>Coupling within the class hierarchy is increased because all classes are directly coupled to the same table. A change in one class can affect the table which can then affect the other classes in the hierarchy.</td>
</tr>
<tr>
<td></td>
<td>Easy to add new classes, you just need to add new columns for the additional data.</td>
<td>Space potentially wasted in the database.</td>
</tr>
<tr>
<td></td>
<td>Supports polymorphism by simply changing the type of the row.</td>
<td>Indicating the type becomes complex when significant overlap between types exists.</td>
</tr>
<tr>
<td></td>
<td>Data access is fast because the data is in one table.</td>
<td>Table can grow quickly for large hierarchies.</td>
</tr>
<tr>
<td></td>
<td>Ad-hoc reporting is very easy because all of the data is found in one table.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Easy to do ad-hoc reporting as all the data you need about a single class is stored in only one table.</td>
<td>When you modify a class you need to modify its table and the table of any of its subclasses. For example if you were to add defaultValue to the Indicator class you would need to add columns to the AreaIndicator and PowerIndicator tables.</td>
</tr>
<tr>
<td></td>
<td>Good performance to access a single object’s data.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Easy to understand because of the one-to-one mapping.</td>
<td>There are many tables in the database, one for every class (plus tables to maintain relationships).</td>
</tr>
<tr>
<td></td>
<td>Very flexible and allows you to keep adding more derived classes without impacting any of the existing code in your application.</td>
<td>Loading a derived class involves a join of multiple tables (although the join is on primary keys) and is therefore a little slower.</td>
</tr>
<tr>
<td></td>
<td>Very easy to modify superclasses and add new subclasses as you merely need to modify/add one table.</td>
<td>Ad-hoc reporting on your database is difficult, unless you add views to simulate the desired tables.</td>
</tr>
<tr>
<td></td>
<td>Data size grows in direct proportion to growth in the number of objects.</td>
<td></td>
</tr>
</tbody>
</table>

1 – One table per hierarchy
2 – One table per concrete class
3 – One table per class
12.2 PRODUKTIV+ Class Model (based on ontology version 1.6)
12.3 Relational Data Model I
12.4 Relational Data Model II
### 12.5 T-HIT Test report

<table>
<thead>
<tr>
<th>Section Id</th>
<th>Driver</th>
<th>Start</th>
<th>Stop</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ConfigProject</td>
<td>22.08.2006 11:32</td>
<td>22.08.2006 11:32</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Vimport</td>
<td>22.08.2006 11:33</td>
<td>22.08.2006 11:33</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Vimport</td>
<td>22.08.2006 11:33</td>
<td>22.08.2006 11:33</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>CreateUnit</td>
<td>22.08.2006 11:33</td>
<td>22.08.2006 11:33</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Workaround</td>
<td>22.08.2006 11:33</td>
<td>22.08.2006 11:34</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Workaround</td>
<td>22.08.2006 11:34</td>
<td>22.08.2006 11:34</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Mbist</td>
<td>22.08.2006 11:34</td>
<td>22.08.2006 11:36</td>
<td>125</td>
</tr>
<tr>
<td>8</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 11:35</td>
<td>22.08.2006 11:36</td>
<td>83</td>
</tr>
<tr>
<td>9</td>
<td>Edt_ip</td>
<td>22.08.2006 11:36</td>
<td>22.08.2006 11:37</td>
<td>43</td>
</tr>
<tr>
<td>10</td>
<td>Vimport</td>
<td>22.08.2006 11:37</td>
<td>22.08.2006 11:37</td>
<td>262</td>
</tr>
<tr>
<td>11</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 11:37</td>
<td>22.08.2006 11:37</td>
<td>223</td>
</tr>
<tr>
<td>12</td>
<td>SpyGlassRTL</td>
<td>22.08.2006 11:41</td>
<td>22.08.2006 11:43</td>
<td>111</td>
</tr>
<tr>
<td>13</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 11:43</td>
<td>22.08.2006 11:44</td>
<td>88</td>
</tr>
<tr>
<td>14</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 11:44</td>
<td>22.08.2006 11:46</td>
<td>81</td>
</tr>
<tr>
<td>15</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 11:46</td>
<td>22.08.2006 11:49</td>
<td>214</td>
</tr>
<tr>
<td>16</td>
<td>SpyGlassRTL</td>
<td>22.08.2006 11:49</td>
<td>22.08.2006 11:51</td>
<td>98</td>
</tr>
<tr>
<td>17</td>
<td>CVE</td>
<td>22.08.2006 11:51</td>
<td>22.08.2006 11:58</td>
<td>416</td>
</tr>
<tr>
<td>18</td>
<td>CVE</td>
<td>22.08.2006 11:58</td>
<td>22.08.2006 12:09</td>
<td>687</td>
</tr>
<tr>
<td>19</td>
<td>DebussyTDrv</td>
<td>22.08.2006 12:09</td>
<td>22.08.2006 12:11</td>
<td>125</td>
</tr>
<tr>
<td>20</td>
<td>VManager</td>
<td>22.08.2006 12:11</td>
<td>22.08.2006 12:12</td>
<td>37</td>
</tr>
<tr>
<td>21</td>
<td>Specman</td>
<td>22.08.2006 12:12</td>
<td>22.08.2006 12:15</td>
<td>192</td>
</tr>
<tr>
<td>22</td>
<td>PowerTheater</td>
<td>22.08.2006 12:15</td>
<td>22.08.2006 12:18</td>
<td>159</td>
</tr>
<tr>
<td>23</td>
<td>DC</td>
<td>22.08.2006 12:18</td>
<td>22.08.2006 12:27</td>
<td>527</td>
</tr>
<tr>
<td>24</td>
<td>CVE</td>
<td>22.08.2006 12:27</td>
<td>22.08.2006 12:31</td>
<td>239</td>
</tr>
<tr>
<td>25</td>
<td>CVE</td>
<td>22.08.2006 12:31</td>
<td>22.08.2006 12:34</td>
<td>197</td>
</tr>
<tr>
<td>26</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 12:34</td>
<td>22.08.2006 12:35</td>
<td>89</td>
</tr>
<tr>
<td>27</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 12:35</td>
<td>22.08.2006 12:37</td>
<td>96</td>
</tr>
<tr>
<td>28</td>
<td>SpyGlassERC</td>
<td>22.08.2006 12:37</td>
<td>22.08.2006 12:38</td>
<td>59</td>
</tr>
<tr>
<td>29</td>
<td>Fsatpg</td>
<td>22.08.2006 12:38</td>
<td>22.08.2006 12:39</td>
<td>34</td>
</tr>
<tr>
<td>30</td>
<td>Edt_atpg</td>
<td>22.08.2006 12:39</td>
<td>22.08.2006 12:39</td>
<td>26</td>
</tr>
<tr>
<td>31</td>
<td>PrimetimePT_SCRIPTS</td>
<td>22.08.2006 12:39</td>
<td>22.08.2006 12:40</td>
<td>49</td>
</tr>
<tr>
<td>32</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 12:40</td>
<td>22.08.2006 12:41</td>
<td>81</td>
</tr>
<tr>
<td>33</td>
<td>PowerTheater</td>
<td>22.08.2006 12:41</td>
<td>22.08.2006 12:43</td>
<td>119</td>
</tr>
<tr>
<td>34</td>
<td>PrimetimePT_SCRIPTS</td>
<td>22.08.2006 12:43</td>
<td>22.08.2006 12:44</td>
<td>39</td>
</tr>
<tr>
<td>35</td>
<td>PrimetimePT_SCRIPTS</td>
<td>22.08.2006 12:44</td>
<td>22.08.2006 12:45</td>
<td>40</td>
</tr>
<tr>
<td>36</td>
<td>Workaround</td>
<td>22.08.2006 12:45</td>
<td>22.08.2006 12:45</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>Blastfusion</td>
<td>22.08.2006 12:45</td>
<td>22.08.2006 14:43</td>
<td>7083</td>
</tr>
<tr>
<td>38</td>
<td>CVE</td>
<td>22.08.2006 14:43</td>
<td>22.08.2006 14:46</td>
<td>185</td>
</tr>
<tr>
<td>39</td>
<td>Layver</td>
<td>22.08.2006 14:46</td>
<td>22.08.2006 15:30</td>
<td>2649</td>
</tr>
<tr>
<td>40</td>
<td>IremGL</td>
<td>22.08.2006 15:30</td>
<td>22.08.2006 15:34</td>
<td>250</td>
</tr>
<tr>
<td>41</td>
<td>ExtractGL</td>
<td>22.08.2006 15:34</td>
<td>22.08.2006 15:47</td>
<td>778</td>
</tr>
</tbody>
</table>
### Setting up a data gathering infrastructure for PRODUKTIV+

<table>
<thead>
<tr>
<th></th>
<th>Script/Program</th>
<th>Time Stamp 1</th>
<th>Time Stamp 2</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>PrimetimePT_SCRIPTS</td>
<td>22.08.2006 15:47</td>
<td>22.08.2006 15:48</td>
<td>66</td>
</tr>
<tr>
<td>41</td>
<td>PrimetimePT_SCRIPTS</td>
<td>22.08.2006 15:48</td>
<td>22.08.2006 15:49</td>
<td>64</td>
</tr>
<tr>
<td>42</td>
<td>ModelsimTDrv</td>
<td>22.08.2006 15:49</td>
<td>22.08.2006 15:51</td>
<td>95</td>
</tr>
<tr>
<td>43</td>
<td>PowerTheater</td>
<td>22.08.2006 15:51</td>
<td>22.08.2006 15:53</td>
<td>113</td>
</tr>
<tr>
<td>44</td>
<td>Fsatpg</td>
<td>22.08.2006 15:53</td>
<td>22.08.2006 15:53</td>
<td>35</td>
</tr>
<tr>
<td>45</td>
<td>Nanosimdigital</td>
<td>22.08.2006 15:53</td>
<td>22.08.2006 16:12</td>
<td>1105</td>
</tr>
</tbody>
</table>

**Total** 17032
## 12.6 Business Question 1 result

<table>
<thead>
<tr>
<th>Y</th>
<th>M</th>
<th>MP</th>
<th>Project</th>
<th>Tool</th>
<th>License feature</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>projectX</td>
<td>Fsatpg_bf</td>
<td>vhd2gates_tauri</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>projectX</td>
<td>Fsatpg_bf</td>
<td>HDLAMS_TAURI</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>projectX</td>
<td>Fsatpg_bf</td>
<td>gateprop</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>projectX</td>
<td>Fsatpg_bf</td>
<td>rsm2gates</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_VIEW</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_WRAP</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_FUSION_APX</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>gatecheck</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_SI</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_BUILDER</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_NOCV</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>vhd2gates_tauri</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>HDLAMS_TAURI</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_SPEED</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>rsm2gates</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>gatecomp</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_CREATE</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_CREATE_SA</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_FUSION_MVDD_APX</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_FUSION_QT</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_FUSION_SA</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_PLAN</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_PLANPRO</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_PLANsvp</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_POWER</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_RAIL</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_RAIL_NX</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_TEST</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>BLAST_YIELD</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>1</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>QUARTZ_RC</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_VIEW</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_WRAP</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_FUSION_APX</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_SI</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_BUILDER</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_NOCV</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_SPEED</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_RAIL</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_CREATE</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_CREATE_SA</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_FUSION_MVDD_APX</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv</td>
<td>Blastfusion</td>
<td>BLAST_FUSION_QT</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Month</td>
<td>Day</td>
<td>Product</td>
<td>Feature</td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>-----</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_FUSION_SA</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_PLAN</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_PLANPRO</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_PLANVP</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_POWER</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_RAIL_NX</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_TEST</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>BLAST_YIELD</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Blastfusion</td>
<td>QUARTZ_RC</td>
<td>468</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn</td>
<td>gatecheck</td>
<td>3534</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn</td>
<td>vhdl2gates_tauri</td>
<td>372</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn</td>
<td>HDLAMS_TAURI</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn</td>
<td>HDLAMS_TAURI</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn</td>
<td>vhdl2gates_tauri</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn wo</td>
<td>HDLAMS_TAURI</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn wo</td>
<td>vhdl2gates_tauri</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn wo</td>
<td>verilogRTL2gates</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn wo</td>
<td>HDLAMS_TAURI</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_RtlvsSyn</td>
<td>vhdl2gates_tauri</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_SynvsBf</td>
<td>rsm2gates</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_SynvsBf</td>
<td>verilogRTL2gates</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_eqcheck_SynvsBf</td>
<td>gatecomp</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_IntentChecking</td>
<td>rsm2gates</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_IntentChecking</td>
<td>gateprop</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_IntentChecking</td>
<td>gatecomp</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_PropertyChecking</td>
<td>gatecomp</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_PropertyChecking</td>
<td>verilogRTL2gates</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_PropertyChecking</td>
<td>rsm2gates</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv CVE_PropertyChecking</td>
<td>rsm2gates</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Layver_bf</td>
<td>netlistHerc</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>2</td>
<td>produktv Project Setup</td>
<td>gateprop</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### 12.7 Business Question 2 result

<table>
<thead>
<tr>
<th>Y</th>
<th>M</th>
<th>D</th>
<th>Project</th>
<th>Tool</th>
<th>Task</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>bind</td>
<td>2158</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>ocv</td>
<td>679</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>gocv</td>
<td>531</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>wire</td>
<td>523</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>cell</td>
<td>450</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>final</td>
<td>326</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>clock</td>
<td>290</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>time</td>
<td>252</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>pt_corr</td>
<td>225</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>power</td>
<td>217</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>partition</td>
<td>198</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>plan</td>
<td>181</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>rail_em</td>
<td>133</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>export</td>
<td>127</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>rail_vd</td>
<td>95</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>bfusion</td>
<td>sdc_conv</td>
<td>84</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>calibre</td>
<td>run</td>
<td>9927</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>edt_atpg</td>
<td>run</td>
<td>0</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>edt_ip</td>
<td>run</td>
<td>0</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>fsatpg</td>
<td>fs_run</td>
<td>0</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iremGL</td>
<td>run</td>
<td>101</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iremGL</td>
<td>setup</td>
<td>8</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwdc</td>
<td>compile</td>
<td>265</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwdeb</td>
<td>compile</td>
<td>3</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwdeb</td>
<td>run</td>
<td>3</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwequiv</td>
<td>compile</td>
<td>64</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwequiv</td>
<td>extract</td>
<td>43</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwequiv</td>
<td>extract_revised</td>
<td>25</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwequiv</td>
<td>compile_revised</td>
<td>20</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwequiv</td>
<td>run</td>
<td>20</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwintent</td>
<td>run</td>
<td>702</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwintent</td>
<td>extract</td>
<td>15</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwmsim</td>
<td>run</td>
<td>258</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwmsim</td>
<td>compile</td>
<td>27</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwmsim</td>
<td>msimcompile</td>
<td>4</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwns</td>
<td>runsim</td>
<td>16736</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwprop</td>
<td>run</td>
<td>93312</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwprop</td>
<td>extract</td>
<td>15</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwpt</td>
<td>run</td>
<td>69</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwpt</td>
<td>annotate</td>
<td>44</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwpt</td>
<td>compile</td>
<td>29</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwsdccheck</td>
<td>sdccheck</td>
<td>0</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwsn</td>
<td>run</td>
<td>13</td>
</tr>
<tr>
<td>2006</td>
<td>8</td>
<td>22</td>
<td>produktv</td>
<td>iwsn</td>
<td>genstubs</td>
<td>3</td>
</tr>
<tr>
<td>Date</td>
<td>Time</td>
<td>Username</td>
<td>Command</td>
<td>Result</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-----------</td>
<td>------------------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>iwspy</td>
<td>compile</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>iwvman</td>
<td>run</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>mbist</td>
<td>run_mbist</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>mgcesd</td>
<td>run</td>
<td>9927</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>ptiw</td>
<td>auto</td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>ptiw</td>
<td>writesdf</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>ptiw</td>
<td>ptiw_ptCorr</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>StarXtract</td>
<td>run</td>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006 8 22</td>
<td>produktv</td>
<td>v2lvs</td>
<td>run</td>
<td>9927</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>