Software Construction

RWTH AACHEN UNIVERSITY

Faculty of Mathematics, Computer Science, and Natural Sciences

Research Group Software Construction

Master Thesis

Tool-Supported Project Prediction

Werkzeugunterstütze Projektprognose

submitted by

Elena Emelyanova

born in Belgorod

on May 5, 2015

Reviewers

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisors

Dipl.-Inform. Matthias Vianden

Dipl.-Inform. Andreas Steffens

I, Elena Emelyanova, hereby affirm in lieu of oath that the Master Thesis at hand is my own written work and that I have used no other sources and aids other than those indicated. All passages quoted from publications or paraphrased from these sources are properly cited and attributed.

The thesis was not submitted in the same or in a substantially similar version, not even partially, to another examination board and was not published elsewhere.

Aachen, May 5, 2015 (Elena Emelyanova)

Acknowledgment

First and foremost, I would like to thank Prof. Dr. rer. nat. Horst Lichter for offering me the opportunity to carry out my thesis. Additionally, I would like to thank Prof. Dr. rer. nat. Bernhard Rumpe for reviewing this thesis.

I am very grateful to Matthias Vianden and Andreas Steffens for being my supervisors, who have guided me with useful comments, feedback, and engagement thoughout my thesis.

Many thanks go to the team of the SWC Research Group of RWTH Aachen University for establishing a friendly and cheerful atmosphere to work in.

I would like to express an special thank you to Andrej Dyck for patient reading my thesis, asking me challenging questions, and giving me valuable advices. Also I would like to thank him for providing many laughs, insightful discussions, and sharing a few beers with me. Thanks for keeping me going during my hard work.

Last but not the least, I am indebted to my family and all my friends who have supported me during my studies.

Elena Emelyanova

Abstract

Dealing with an increasing complexity of software, and thus, software projects, managing those projects become more and more difficult. While monitoring a software project is a necessary task to identify potential failures or quality loss, often projects managers can only react. On the other hand, predicting the project's progress can help project managers to avoid certain pitfalls. For this purpose, they often rely on their experience. In order to support unexperienced managers or help in new situations, a tool-based prediction using historical project data were proposed.

To the best of our knowledge, all proposed prediction methods deal with metrics such as lines of code, number of defects, etc. As project managers are usually more interested in metrics such as CPI and SPI, in this thesis, we contribute with a framework that facilitates prediction for all kinds of metrics. This framework uses data mining methods to gain knowledge from past projects, and with that, predict the progress for an ongoing project. Moreover, we implemented a prediction tool that forecasts CPI and SPI values, and evaluates and visualizes the prediction results.

We evaluated the prediction tool with the historical data of ITERGO. The evaluation shows promising results. Moreover, the framework is designed for adaptability and extendability. Each knowledge discovering step is encapsulated in a single component with a clear API. Moreover, the visualization as well as the data mining library are loosely-coupled and can be easily replaced. Thus, enabling the framework to be used in different problem domains.

Contents

1 Introduction **1**
- 1.1 Objectives . . . 1
- 1.2 Structure of this Thesis . . . 2

2 Background **5**
- 2.1 Software Project Management . . . 5
- 2.2 Frameworks . . . 8
- 2.3 Data Mining . . . 12

3 Requirements **19**
- 3.1 Stakeholders and Requirements of the PredictionTool . . . 19
- 3.2 Stakeholders and Requirements of the DataMiningFramework . . . 22

4 Related Work **25**
- 4.1 Software Project Schedule Prediction Using Bayesian Network . . . 25
- 4.2 Prediction of Defects in Software Projects . . . 26
- 4.3 Discussion . . . 36

5 Concept **37**
- 5.1 Idea of the Framework . . . 37
- 5.2 Processes Modeling . . . 38
- 5.3 GUI Prototyping . . . 46
- 5.4 Domain Model . . . 50

6 Realization **53**
- 6.1 JavaEE Technologies . . . 53
- 6.2 Architecture . . . 55
- 6.3 WEKA Java API . . . 64
- 6.4 Visualization . . . 71
- 6.5 Evaluation of Clustering Performance . . . 73

7 Evaluation **79**
- 7.1 Evaluation of the DataMiningFramework . . . 79
- 7.2 Evaluation of the PredictionTool . . . 82

8 Conclusion **99**
- 8.1 Summary . . . 99

8.2 Future Work 100

9 Appendix **103**
9.1 Experiment 1: Box Plot Diagrams 103
9.2 Experiment 2: Box Plot Diagrams 103

Bibliography **113**

1 Introduction

The increasing complexity of software projects is making the management of those projects more difficult. While planning software development, managers need to take in account a variety of factors; including effort requirements, project costs, defect density, schedule delay, etc. Usually, managers have to make decisions regarding future project's events based on their personal experience, which is often problematic or impossible due to missing experience. With the objective to support mangers in decision making, several prediction techniques and tools for a project's progress have been proposed. Still, a reliable prediction is one of the most challenging issue that concerns software project managers and is still underexplored.

Most research suggests prediction methods for project metric values based on historical data, i.e., of past projects, as well as the experience of specialists. In other words, the information from past projects, namely a collection of measured metrics, is used to derive metric value trends for an ongoing project. To this end, the information is differentiated and classified according to certain criteria. Here, often data mining techniques like clustering and classification are applied. Then, the ongoing project is assigned to one of the classes, and with that, the progress is assumed to be similar to the characteristics of this class.

To the best of our knowledge, all of the proposed methods deal with metrics such as lines of code, number of bugs, etc. However, project managers are usually more interested in metrics such as CPI and SPI, and thus, in the prediction of a project's behavior. In this thesis, we tackle the project metric values prediction of any kind by proposing a framework of data mining steps that can be used within various problem domains. Additionally, as a proof of concept, we present a tool to predict CPI and SPI values for an ongoing project based on historical data. With this, we support project managers in their decision making.

1.1 Objectives

The endeavor of this thesis is to develop framework for automatic prediction of future project's metric values. Thereby, we focus on designing a forecasting system that is based on the collaboration of software engineering solutions and data mining methods. The system calculates a prediction based on various performance evaluation metrics, e.g., CPI, SPI, the total number of tickets, etc. The metrics are transformed into discrete values that compose the input of the system. The framework must facilitate the transformation of the discrete values into time series that are used for a further data mining processing. Consequently, the framework must provide a data mining

component that is needed to identify similarity patterns in time series for prognosis creation. For further analysis, the results obtained from the data mining component, must be processed by the visualization component of the framework.

The forecasting framework must enable the user to apply various computing components to different time series in order to discover data knowledge. To specify the goals of of this thesis, we summaries the objectives of this work as follows:

Performing of practice-oriented research to determine whether a similar 'best practice' method for meaningful clustering of time series already exists. Based on this research, we want to identify which regression and clustering algorithms can be selected for the core steps of the forecasting tool.

Designing a component-based data mining framework that offers solutions for specific problem domains. The framework must provide a loosely-coupled architecture, where each step of the data mining processing is encapsulated by a single component. Furthermore, the framework must be open for the extendability and changeability of components by providing extension points. Hereafter, we refer to this framework as DataMiningFramework.

Creating a prototype of project prediction tool. The prototype must provide the following functions:

- Import of historical metric data.
- Processing the row data and transformation into the form of time series.
- Providing different clustering methods and their configuration.
- Providing different visualization possibilities of the clustering results.
- Enabling interpretation and evaluation of clustering results.

Hereafter, we call the prediction tool the PredictionTool. Here, the PredictionTool serves as a proof of concept of the DataMiningFramework

Evaluating the approach with industry data. This objective includes evaluation and analysis of different regression and clustering algorithms based on the empirical database of the company ITERGO.

1.2 Structure of this Thesis

This thesis is divided into the following seven chapters:

Chapter 2 - Background This chapter addresses the theoretical foundations of this thesis. Since our prototype targets a software project manager's needs, we start by explaining the term of software project management in section 2.1. Thereby, we define the terms of project life-cycle and earned value analysis (see sections 2.1.1 and 2.1.2). Then, in section 2.2, we explain essential terms, characteristics, and types of software frameworks. Last, in section 2.3, we outline the data mining approach and its important terms and definitions.

Chapter 3 - Requirements In this chapter, we formulate the capabilities and properties that must be provided by our prototype. Thereby, we identify the stakeholders and derived the requirements from their needs. Additionally, we differentiate the stakeholders and their requirements into two groups: the stakeholder needs and requirements for the PredictionTool in section 3.1 and for the DataMiningFramework in section 3.2.

Chapter 4 - Related Work In this chapter, we present an overview of previous work on prediction of software project progress. Thereby, we list the a number of related papers and discuss their results as well as the pros, cons, and areas of improvement in context of project prediction. Moreover, we emphasize the contributions of our work.

Chapter 5 - Concept This chapter describes the main idea and details of the software prototype designed in this work. First, we concentrate on the main components of the whole prediction process in section 5.1. After that, we present the process model and explain the single activities and processes of the system by using flow diagrams in section 5.2. Then, in section 5.3, we design a GUI prototype for the PredictionTool. Finally, we present the domain model of the system's architecture in section 5.4.

Chapter 6 - Realization Subsequently, in chapter 6 we provide the implementation details of the prototype. Thereby, we start with describing the JavaEE techniques and tools used for the realization task in section 6.1. After that, we present the component architecture of the prototype and describe its details in section 6.2. Furthermore, we describe the integration details of the data mining tool used in the computation core in section 6.3. Finally, we lay out the visualization techniques that are used in our prototype in section 6.4.

Chapter 7 - Evaluation This chapter describes evaluation methods and their results used to estimate the performance of the prototype. Thereby, we split the chapter into two main parts. In the first part, we present the techniques used to evaluate DataMiningFramework and discuss the obtained results. In the second part, we describe several experiments that we designed and performed to evaluate the PredictionTool.

Chapter 8 - Conclusion Finally, we summarize and discuss the results of this thesis in section 8.1 and give an outlook on future work in section 8.2.

2 Background

Contents

2.1 Software Project Management 5
2.1.1 Project Life-cycle 6
2.1.2 Earned Value Analysis 6
2.2 Frameworks 8
2.2.1 White-box and Black-box Frameworks 9
2.2.2 Hook and Template Methods 11
2.3 Data Mining 12
2.3.1 Supervised and unsupervised learning 13
2.3.2 Regression Analysis 14
2.3.3 Clustering Analysis 14
2.3.4 Weka Data Mining Software 17

The endeavors of this thesis address the development of a data mining framework with the main purpose to help extracting information from the data set of completed projects and predict development progress of ongoing projects. To this end, we first briefly recall terms and definitions of *software project management* in section 2.1. In section 2.2 we explain the main characteristics and special futures of the architecture of software frameworks. The last section of this chapter 2.3 handles the data mining topic, where we give background information and theoretical foundations on *data mining* analysis techniques.

2.1 Software Project Management

In this section we present software project management and describe the key characteristics of a project and the project management process. The term project management is closely linked to the term project. In literature exist several definitions for the term project. We choose the definitions of project proposed by Kerzner and the Project Management Institute (PMI) in [Ker09] and [PMI04], respectively. According to the both definitions, the project term can be summarized as a *"sequence of organizing activities and tasks which have to be performed according to a certain specification and constrain in order to create a unique product or service."* Each project has an objective, as well as a start and an end date. Furthermore, certain amount of human and non-human resources are assigned to each project.

In order to introduce the term software project, we use the definition proposed by Thayer, explaining software project as a project with the objective to develop a

software system [You00]. Ludewig and Lichter define in [LL10] further characteristics of software project; namely, the presence of project owner and project purchaser, as well as the establishment of relationship between staff, a projects' output, and the projects' resources.

PMI defines software project management as *"the application of knowledge, skills, tools and techniques to project activities to meet project requirements"* [PMI04]. Thayer outlines five main functions of the project management process: planning, organizing, staffing, directing and controlling. Planning and organizing comprise activities that are needed to complete organizational objectives, e.g., developing strategies and polices, preparing specification of procedures, rules and documentation, forming teams and defining responsibilities of their members. Staffing and directing deal with hiring and motivating the staff who performs the project activities. Controlling covers evaluation of the performance of the project activities. A detailed explanation of each function and its activities can be found in [You00, Ker09].

2.1.1 Project Life-cycle

In order to reduce managerial effort and improve management control, software projects are typically divided into several project phases. In general, the composition of project phases forms the project life-cycle. Each phase has a predefined start and end dates, where the latter is assigned to a milestone within the project. Milestone can be defined as a set of phase objectives. A phase is considered successful completed if a project phase ends within its deadline and satisfies all criterions defined for the corresponding milestone. As a consequence, a milestone helps to represent an progress point of the project development process, and supports the examination task. The activity of splitting a project into phases should be performed carefully. In practice, too short as well too long phases negatively affect the project development process [LL10].

There exists no general agreement on how to organize a projects' lifecycle. However, PMBOK®Guide defines five theoretical life-cycle phases of a project, which are conceptual, planning, testing, implementation, and closure [PMI04]. In software project management, this concept is extended through various software process models, e.g., Waterfall model, iteration model, Spiral model, etc. A software process model is used to define a software development process and its phases in an abstract manner. Each software process model typically include analysis, specification, design, implementation, testing, evaluation, and maintenance phases. For more information see [Ele90, Som06, LL10].

2.1.2 Earned Value Analysis

Project control includes monitoring and reporting the work performance of project activities according to its scope, schedule, costs, and risks. For instance, the Earned Value Analysis (EVA) is an approach to monitor and control of project costs performance. EVA has been originated by the U.S. government in 1967 to be integrated into government cost and scheduler control systems. Due to its success, it was adapted by

private industrial companies and set the standard for measurement project performance against its costs and schedule [FK02].

The earned value technique is a key performance indicator of a project w.r.t. earned value management. The EVA is performed for a specific task or work at a certain point of time. The results are used to prognosticate total project costs and date of project completion. EVA is based on the variance of the three main key values: *earned value* (EV), *planned value* (PV), and *actual cost* (AC). Earned value addresses the amount of the work actually performed during a given time period. Planed value describes the budgeted costs of the work according to the project schedule up to a given point in time. Actual costs are the total effective costs resulting from the performed work on the scheduled activity. Accordingly, the EV, PV, and AC measurements are used to determine whether a project task is completed according to project schedule or not. For this purpose, cost and schedule variance are computed as follows:

- **Cost variance:** $CV = EV - AC$
- **Schedule variance:** $SV = EV - PV$

Cost variance provides information whether a project is under or over budget. Here, a negative value of cost variance indicates that project is over budget for the work performed. The positive value of CV refer to the project under budget and the CV equal to null indicates that the cost spent for the work corresponds the planned value. Schedule variance shows whether a project is ahead of or behind schedule. In particular, a negative value of schedule variance hints that the planned work has not been completed and a positive value indicates a favorable state of project.

While cost and scheduler variance are absolute values, *cost performance index* (CPI) and *schedule performance index* (SPI) represent project performance as relative values. Here, CPI helps to estimate the cumulative cost efficiency of a project and indicates how efficiently the project resources has been using. CPI is computed using EV and AC:

$$CPI = \frac{EV}{AC}$$

Typically, the CPI value, which is within interval $[0, 1)$, indicates unfavorable conditions of projects' budget, i.e., spent more than planned. The CPI value that is equal to one implies that the cost for completing a task or work is less than initially planned. Consequently, the CPI value that is equal to one indicates that the costs spend correspond to the planned value.

Scheduler performance index (SPI) shows how efficiently the project time is used and is computed using EV and PV:

$$SPI = \frac{EV}{PV}$$

The SPI value determines that on average for each 8-hour day the percentage of time used to perform the work. The SPI value, which is within interval $[0, 1)$, indicates inefficient usage of working time (performed less than planned). SPI that is nearly or equal to 1 shows that the project is in time, otherwise if the SPI value is grater than one mens that it is performed more than initially planned [Ins04, Ker09].

2.2 Frameworks

The construction of large-scale software systems often rely on reusable *software components*, since the application of already implemented and tested parts of a program significantly increase reliability. Furthermore, the software development that is based on a component-based approach lowers implementation efforts and maintenance costs [BR90].

For component-based software development, there exists several definitions trying to explain the meaning of a software component. Intuitively, a software component is conceived as an independent and replaceable software part that encapsulate a certain functionality. Typically, each component provides a predefined interface and conforms to an established behavior in the software system. In the IEEE Standards Definition a software component is defined as a *"general term used to refer to a software system or an element, such as module, unit, data, or document"* [faSQMM15]. Szyperski explains the term of software component as a *"unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition."* [SGM02]. Here, the important characteristic is the separation of interfaces and their implementation.

In contrast to reusing a single component, a more effective reuse is one of a pre-defined software architecture, a so-called *framework*. Framework is collection of software components with the pre-defined interaction between them [FPR00]. Hence, the term component-based development is closely-related to the term framework-based development. Thereby, frameworks can be applied for developing software components by implementing an interface to access the internal framework's architecture. Another way is to use the framework as a black-box, i.e., the components of the frameworks are used as pluggable ready-to-use components. The common application of the framework concept is to enhance and facilitate the development of infrastructure and middleware software. The component-based approach supports end-user software development [FS97].

Pree defines a framework as a *"reusable semifinished architectures for various application domains. Such frameworks represent a real breakthrough in software reusability: not only single building blocks, but also the design of (sub)systems can be reused"* [Pre95c]. The framework concept enhances past reuse techniques, such as methods that are based on class libraries and introduces the new approach of semi-complete application for particular business units and application domains. Figure 2.1 reflects the difference between function calls in the application that uses several libraries (a) and framework (b). Opposite to an application involving class libraries, a framework provides the crucial library components by itself and calls back specific application procedures and functions [FPR00].

Application of a framework in software development carries with it several advantages regarding the quality of software; namely, it targets to improve modularity, reusability, extendability, and implementing inversion of control [FS97]. The terms can be explained as follows:

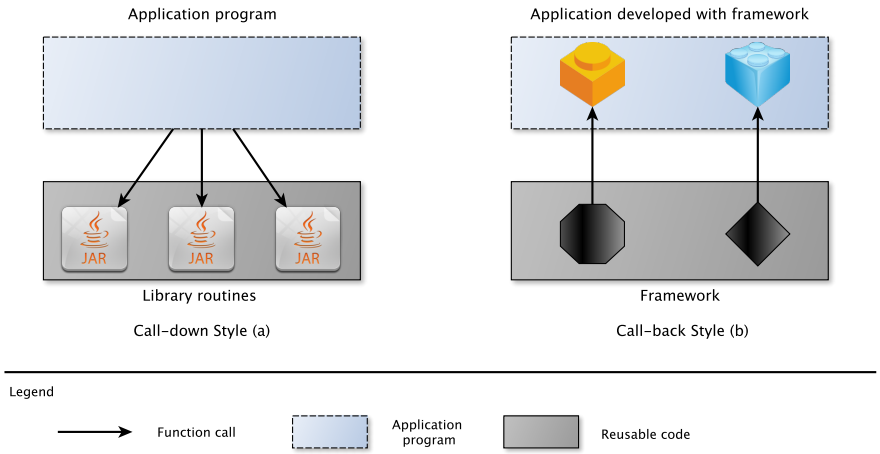

Figure 2.1: Call-down (a) and call-back (b) style of programming [FPR00]

Modularity A better modularity of the system is achieved by frameworks due to encapsulating the functionality behind interfaces. A modular system is broken into self-contained modules that are developed separately without affecting other modules. The latter issue lows the effort that is needed to comprehend and maintain existing software.

Reusability A framework's architecture defines components that are designed and implemented with the objective to be reused in the other applications. The motivation behind the concept of reusability is avoiding re-creation of common solutions and using instead pre-defined components or architecture that are designed based on previous experience.

Extensibility A framework's architecture provides a further crucial function: namely, extendability of the software, i.e., it specifies extendable interfaces that offer adoption for new features. This extensibility ensures customization of new application services and functionality.

Inversion of control The classical call-down principal defines hard-wired interaction between components that excludes any dynamic alteration of code. In contrast, frameworks inverse the control of the program's flow that is not dictated by the framework caller, but by the framework. Inversion of control allows a component to decide at the run time which dependencies have to be injected.

2.2.1 White-box and Black-box Frameworks

Frameworks allow variation of functionality for a particular software application by providing extension points or specialization. Pree defines a framework as *"a high level language"* that enables creation of reusable software solution through specialization using

hot-spots [Pre95a]. Moreover, Pree explains two types of specialization techniques: the specialization performed through composition that is called *black-box*, and specialization conducted by using inheritance that is defined as *white-box* [Pre95b].

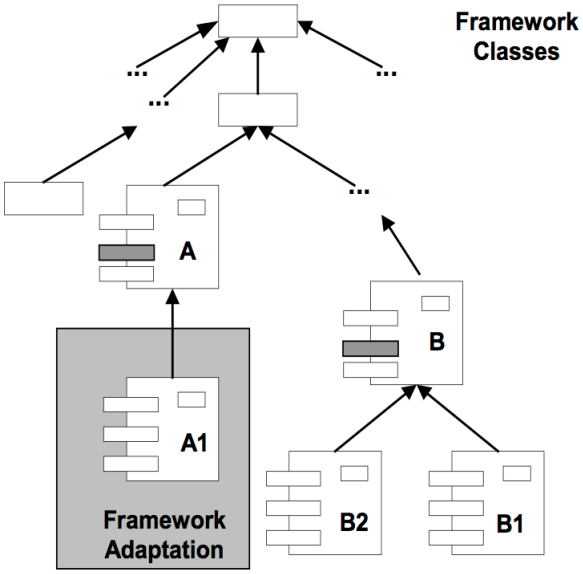

Figure 2.2: Framework class hierarchy [Pre95b]

Hot-spots are either abstract classes or interfaces that provide an abstract implementation of features, and so, offer the adaption of features through a concrete implementation in a certain application. The concept of hot-spots enables standardization of the application structure and defines collaboration behavior between components inside a framework.

Figure 2.2 illustrates an example of a white-box framework architecture with a pre-defined adoption point, i.e., a hot-spot: the gray rectangle of class A, which, here, is an abstract method. Here, the method of superclass A is overridden by the subclass $A1$ by applying inheritance mechanisms. An alternative to the methods where the abstract method is applied is using of an interface. In this case, any class that implements the interface A can is used to adapt the framework.

Consequently, white-box components are semi-complete classes that contains methods with default implementation (hot-spots), which must be rewritten by a concrete adaptee class. This requires the developer to understand the architecture and implementation of the framework, and implies a good documentation of framework's extension points.

In contrast to white-box, the black-box approach of a framework defines ready-to-use components. In this case, hot-spots are implemented by composition and parameter definition. Figure 2.3 (a) demonstrates a black-box specialization with two extension points A and B of a component. The class of type B is an abstract class that has two concrete implementations (subclasses), $B1$ and $B2$. The adaption is performed by

instantiating classes $A1$ and $B2$ and plugging in the corresponding objects (cf. figure 2.3 (b)). The developer can directly use class B for the adaption, while in the case of class A, a concrete subclass $A1$ must be created first.

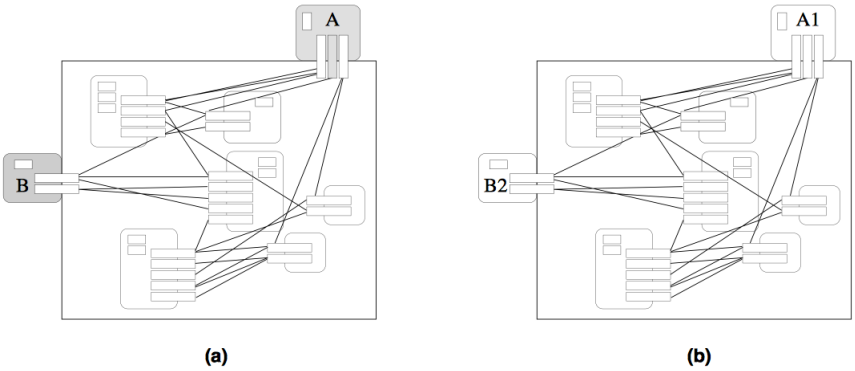

Figure 2.3: Framework: Balck-box approach before (a) and after (b) specialization [Pre95b]

In practice, frameworks are neither designed as pure a black-box, nor as a pure white-box framework. Typically, it is designed as a combination of both component types. However, black-box components facilitate reusability of framework. Hence, the framework's maturity is reflected by the quota of black-box components.

2.2.2 Hook and Template Methods

The hot-spot-driven framework development defines two types of hot-spot methods, namely, the *hook* and the *template* methods. Hook methods are typically abstract methods, i.e., they have no implementation, and serve as place holder (hooks) for a particular implementation. Hook methods are invoked then by the more complex methods that are called templates. Templates are methods that set abstract behavior of objects as well as interaction between them. The hook methods enable modification of object's behavior without modification of the corresponding class by overriding hooks through inheritance [Pre95b].

Figure 2.4 (a) demonstrate invocation of a hook method by the template method. Since the hook method is suppose to be an abstract method it must be overridden in a subclass $A1$ implementing abstract class A (see figure 2.4 (b))

In practice, hook and template methods are separated in two classes. The class that contains hook method is called *hook class* and the class containing template method is denoted as *template class*. Typically, the template class is represented through an abstract class (or interface) that is extended (or implemented) by the hook class. Separation of template and hook class through a inheritance relationship provides better software modularity and extendability. Furthermore, separation of template and hook classes offers run-time adaptation which implies that the same template method can

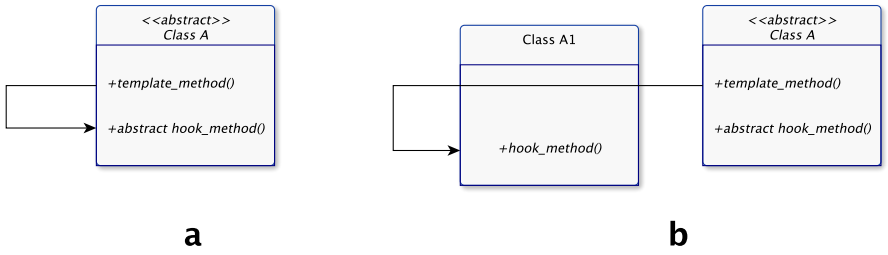

Figure 2.4: Template and hook methods (a) and hook overriding (b) [Pre95b]

invoke multiple hook methods during the system execution.

2.3 Data Mining

A key aspect of data mining is prediction of behavior, performance, or properties if subjects. Hence, cluster analysis can be used to identify groups, and thus, draw conclusions for a single subject. In this section, we explain the term data mining first. After that we discuss the techniques used in data mining and define essential terms related to the data mining area. Finally, we present various clustering methods as well as a prediction method using data mining.

Data mining comprises a big range of techniques and methods that are applied to large amounts of data in order to extract patterns or structure. Kamber et al. [HK00] defines the term data mining as *"extracting or mining knowledge from large amounts of data [..] mining is a vivid term characterizing the process that finds a small set of precious nuggets from a great deal of raw material"*. The term data mining is often understood as a synonym for *knowledge discovery from data*, or *KDD*.

The knowledge discovery process can be explained as a iterative sequence of the following steps:

1. Data selection where only the data necessary for the analysis is chosen
2. Data pre-processing
3. Data transformation in representation form needed for analysis
4. Data mining where different methods are applied to discover patterns from data
5. Pattern evaluation and representation of obtained knowledge

First, data selection filters the data considering the importance for the current analysis. Then, the data pre-processing task is responsible for cleaning data from noises and inconsistencies. Third, data transformation is performed in order to convert data into the input format required by the data mining step. This step includes various methods to discover patterns in the data. Last, the gained knowledge is represented by applying

various visualization and knowledge representation techniques to enable interpretation of the result.

The data input in data mining takes the form of *instances* and *attributes*. An instance is an individual, independent peace of information that is typically associated with the database typles represented by an n-dimensional attribute vector. Each instance is characterized by one or more attributes that describes different aspects of the instance. There are many different types of attributes, e.g., numeric, nominal, categorical, etc. A set of instances forms the data input. In data mining we distinguish between *training* and *test sets* of instances. The former is set of instances that is used to build clusters. The test set is a set of "unseen" instances, i.e., the test instances that are not included in the train set neither during model training nor during model usage.

2.3.1 Supervised and unsupervised learning

Data mining techniques can be separated into two large groups: *supervised* and *unsupervised learning*. Supervised learning refers to data mining algorithms that are applied to extract model containing important data classes, and is a synonym for *classification*. Data classification is performed in two steps: First, we have to built a classifier, or classification model, from the training set. Second, this model is used for to classify the test set. Subsequently, we will describe both steps in more detail.

The first step, called *training step*, aims to describe a predetermined set of data classes. To this end, a classification algorithm creates the classifier by analyzing the training set. During classification it is assumed that each instance in the training set belongs to a to a predefined class, i.e., the classification process requires that the class label of each training instance is known before the classifier is built (this implies that the learning of the classifier is *"supervised"*). The class label is represented by special attribute of instance, called *class label attribute*.

In the second step, the classifier obtained in the first step is used for classification, i.e., assign instances to classes. This step operates the test set of instances that is independent (unknown) from the train set. The resulting accuracy of the classification model (or classifier) on a given test set is computed from the percentage of test set instances that are correctly classified by the classifier.

Another supervised learning process is *prediction*. In contrast to classification, prediction does not apply a class label attribute to instances. Instead, it builds a prediction model, first, and then, uses this model to predict *continuous-valued* or *ordered* attributes for the test set. The accuracy of a prediction model is computed from the difference between the predicted values and the (latter) known value.

Clustering, on the other hand, is a technique of unsupervised learning. In this case, the number of classes and their labels for the training instances are unknown. This type of learning belongs to *learning by observation*. Kamber defines clustering as *"the process of grouping a set of physical or abstract objects into classes of similar objects"* [HK00]. By doing so, a number of clusters is generated. Then, each cluster is characterized as a collection of similar objects that are dissimilar to the objects in other clusters. There are

a number of algorithms used for unsupervised learning. We integrated several clustering algorithms in this thesis for the evaluation purpose, which are described in section 2.3.3.

2.3.2 Regression Analysis

Regression analysis is a statistical method to model the relationship between independent and dependent continuous valued variables. Typically, independent and dependent variables are called predictors and response variables, respectively. The simplest form of regression analysis is *linear regression*, where the response variable y is a linear function of the predictor x. The general linear regression model in 2-dimensional space is given by

$$y = a_0 + a_1 x,$$

where a_0 and a_1 are constant regression coefficients that define the intercept and slop of the resulting regression line. The constant coefficients can be determined with the method of least squares: For a training set with n objects of the form $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, a_0 and a_1 is computed as follows:

$$a_0 = \overline{y} - a_1 \overline{x} \text{ and } a_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2},$$

where $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ are the average values of $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, respectively.

As we can observe in figure **??**, a linear regression is not always a fitting model and a better approximation is desirable. This can be achieved by applying a non-linear regression analysis; e.g., polynomial regression, where the response variable y is a polynomial function (of degrees larger or equal two) of the predictor x.

2.3.3 Clustering Analysis

In the context of this work, the class labels are initially unknown, and thus, should be determined first by applying unsupervised learning or clustering. Clustering is the process of partitioning similar objects into groups, or so-called clusters. Each cluster is built according to the following characteristics: maximizing the intra-cluster similarity and preserving low inter-cluster similarity. As a result, each object within a cluster has higher similarity, w.r.t. a given similarity function, to the other objects of the same cluster than to objects of other clusters. There exists a vast amount of clustering methods. In this thesis, we focus only on k-Means, Expectation Minimization, density-based, and hierarchical methods. Subsequently, we present those methods

k-Means Clustering

The k-Means clustering algorithm is a partitioning clustering method and is one of the simplest and commonly used clustering approaches. The input parameters of this method are a number k, which defines the amount of clusters, and a set of n elements

that need to be assigned to the k clusters. The elements to be clustered are represented as points $p = (x_1^p, \ldots, x_n^p)$ in an Euclidean n-dimensional vector space. With this, the algorithm tries to assign the elements to the clusters according to a similarity function based on the cluster's mean, or so-called centroid. The centroid μ_C represents a mean value of all points in each cluster C:

$$\mu_C = (\overline{x}_1(C), \overline{x}_2(C), \ldots, \overline{x}_n(C)),$$

where

$$\overline{x}_j(C) = \frac{1}{|C|} \sum_{p \in C} x_j^p$$

is the mean value of all points in the j-th dimension in C.

The k-Means algorithm has five steps:

1. Randomly select k points, which will represents a cluster centroid.
2. Assign the remaining points to the cluster with the closest distance between the point and the cluster mean.
3. Compute the square error E for all clustered points as:
   $$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2,$$
   where p represents a point of cluster C_i and m_i is the mean of cluster C_i.
4. If the square error function does not converge, re-calculate a new mean for each cluster and re-assign points.
5. Repeat steps 2 through 4 until the error function converges and the means of all clusters no longer change.

The algorithm terminates if the square error function converges and no redistribution of points among clusters occurs. The result of this process are k clusters that minimize the square error criterion. The algorithm is efficient with large data sets. However, the algorithm is sensible to noise and outliers, since even small variations in data significantly affect its mean value. Moreover, outliers cannot be identified by partitioning clustering methods, since each element is assigned to exactly one cluster. Hence, clusters obtained by the k-Means algorithm might be non-convex and vary strongly in size. The result of clustering by using k-Means method depends on initial partition of points in the first step of the algorithm [HK00, ES00].

Expectation Maximization

Model-based clustering methods use mathematical models, in order to optimize the clustering process. The Expectation Minimization (EM) method is an example for such a model-based clustering. It is a variance of the k-Means clustering method. In contrast to k-Means, the membership of an element in a cluster is not determent by the distance to another element, but by likelihood of being a member of a particular cluster. Typically,

the Gauß distribution is used to calculate the likelihood. In general, the algorithm operates in two steps: *expectation* and *maximization*. Before the start of the algorithm, an initial model $M' = \{C'_1, \ldots, C'_k\}$ with k clusters C_i is randomly generated.

First, in the expectation step, the probability of membership is computed for each point in a cluster. Each probability that a point p belongs to a certain cluster C_i builds the set of the expected class values. As a consequence, points might belong to more than one cluster with different likelihoods. Then, in the maximization step, the algorithm re-distributes points among clusters, using parameters obtained in the first step, resulting in a new, better matching model $M = \{C_1, \ldots, C_k\}$. Both steps are repeated until the steps yield no changes to the model or a defined number of iteration are performed.

In general, the number of iterations of the EM algorithm is large. Hence, the efficiency of the EM algorithm depends on initial distribution and proper choice of the parameter k. Note that the EM method allows for a clustered point to be a member of multiple clusters in the resulting model. In order to get a definite membership, we can use the highest point-cluster probability as the assignment criteria.

Hierarchical Clustering

The hierarchical clustering method groups data objects by using tree-based data structures, which can be constructed according to bottom-up, *agglomerative hierarchical clustering*, or top-down, *divisive hierarchical clustering*, strategies. The former strategy starts with assigning each object to its own cluster. In the subsequent steps, pairs of similar cluster w.r.t. a similarity criterion are merged into a single cluster until there is only one cluster left or some predefined termination conditions are met. In contrast, the top-down strategy is initiated with one single cluster, containing all objects. The algorithm then splits the clusters until each cluster contains only a single object or some predefined termination conditions are satisfied.

Both, bottom-up and top-down, approaches use a distance function between clusters as similarity criterion. The common distance functions are: minimum distance, maximum distance, mean distance and average distance. Additionally, the user can, for example, define a number of clusters as a further termination criterion, or the distance between clusters exceeds a threashold.

An example for hierarchical clustering is the Single-link algorithm. Single-link uses a threshold distance among objects in a cluster (in top-down) or between clusters (in bottom-up) as a termination criterion. The algorithm operates either in bottom-up or top-down and, in general, has the following steps:

1. Compute the pairwise distances of all objects.
2. Merge the two clusters with the nearest distance (bottom-up) or split a cluster with the widest object distance (top-down).
3. Compute all distances between the new cluster obtained in step 2 and the remaining objects and clusters.

4. Repeat steps 2 and 3 until the termination condition is reached.

The more specific and complete information in terms of clustering and regression analysis, classification, prediction and error estimation can be found in [HK01, HK06, ES00, WF05, HSM01].

2.3.4 Weka Data Mining Software

Waikato Environment for Knowledge Acquisition, or Weka for short, is a Java-based toolkit, providing implementation of various algorithms for data mining and machine learning. It provides a Java API, which allows embedding Weka as a library in applications for data mining tasks. Weka contains packages to enable data preprocessing, clustering, classification, regression analysis and association rule mining. Additionally, Weka offers graphical user interfaces and visualization utilities for algorithm evaluation and analysis.

For this thesis, we use Weka's clustering facilities to determine patterns or classes in the data needed for the prediction task. To this end, Weka provides implementation of several clustering algorithms in the `weka.clusterers` package. Table 2.1 shows the various clustering algorithms and their short descriptions, which are currently part of the Weka data mining tool.

For more information see [BFK+13, Abe10b, Abe10a, HFH+09, Bel14, WF05].

Algorithm Name	Description
Cobweb	A class of clustering functions which implements the Cobweb and Classit clustering algorithms [Fis87, GLF90].
Canopy	A clustering algorithm using the capopy clustering method. The algorithm runs in batch and incremental modes [MNU00].
DBScan	Nearest-neighbor-based clustering algorithm which allows automatically determination of the clusters number.
EM	Implements expectation minimization methods.
FarthestFirst	A clustering algorithm based on the farthest first traversal method [HS85, Das02].
FilteredClusterer	An algorithm which runs an arbitrary clusterer on filtered data. As a consequence, one can apply filters before the clusterer is learned [BFK$^+$13].
HierarchicalClusterer	A clusterer class which Implements a number of hierarchical clustering methods.
MakeDensityBasedCluster	A class which wraps a clusterer to make it return distribution and density.
OPTICS	An extended DBScan algorithm for hierarchical clustering.
sIB	A clustering algorithm based on the sequential information bottleneck algorithm [SFT02].
SimpleKMeans	Implements the simple k-Means algorithm, where the clusters number is specified.
XMeans	An extended k-Means algorithm where the centers are split in its region.

Table 2.1: Clustering algorithms available in WEKA

3 Requirements

Contents

3.1 Stakeholders and Requirements of the PredictionTool 19
3.2 Stakeholders and Requirements of the DataMiningFramework 22

The construction of any software system is doomed to fail without defining requirements, i.e., what the system must and should do, and what constraints it has to satisfy. A *requirement* is a formulation of capabilities and properties of the software system needed by the user to solve a problem [Ele90]. The process of identifying system requirements is one of the crucial and important tasks of software development, since it affects the performance of the whole system in the future. Dorfman et el. states that *"the inability to produce complete, correct, and unambiguous software requirements is still considered the major cause of software failure today"* [DT96].

The requirements identification is based on the user's needs. Technical realization of the system is the task of developer. The latter issue addresses the system constraints that define which tools and techniques are required to design and implement the system. The system user and the system developer are the classical stakeholders of the software system to be constructed.

According to the objectives for this work considered in section 1.1, we differentiate the stakeholders and their requirements into two groups. The first group includes stakeholders and derived requirements for the PredictionTool and the second group addresses the DataMiningFramework. The stakeholders of the PredictionTool are, for example, project manager, project leader, developers. The second group concerning the DataMiningFramework are, for example, project developers, project architects, problem domain experts, etc. Subsequently, we give detailed descriptions for each stakeholder group as well as their requirements.

This chapter is organized as follows: First, we identify the stakeholders and describe their requirements for the PredictionTool in section 3.1. Then, in section 3.2, we do the same for the DataMiningFramework.

3.1 Stakeholders and Requirements of the PredictionTool

This work is based on two main groups of objectives: building the DataMiningFramework that can be applied to generate solutions for different problem domains, and creating the PredictionTool as a a proof of the DataMiningFramework concept. According to this, we identified two groups

of stakeholders with their own specific user needs. In this section we describe the group of stakeholder that are interested in the PredictionTool.

Stakeholders

The stakeholders of the PredictionTool are all persons who use the data mining tools to gather estimation of future project development. Examples for the PredictionTool users are project managers or project leaders whose endeavors are planning, organization, and monitoring ongoing projects. This group of users desire to diagnose potential problems in current projects as early as possible. Hence, they want a reliable prediction of future project's cost and schedule based on historical measurements of already completed projects. Furthermore, the PredictionTool users are most likely no data mining experts. Nevertheless, they should be able to use the PredictionTool.

Further, the PredictionTool users desire to obtain calculations of different predicted metrics in the automatic manner by taking into account historical data. Note that performance and reliability of prediction depends on the amount of the historical data: The more historical data is used to create prediction, the more accurate are the prediction results. Since the data is gathered from the different data sources, e.g., databases, Excel spreadsheet, etc., the PredictionTool users need that the data import methods supports different import options.

Since the PredictionTool users are not required to be the data mining experts, they want to be able trying out different clustering methods in order to see what works for them. Hence, they need that different methods for data preparation and clustering configuration are available. To this end, the configuration should be easy.

The PredictionTool users desire the display of the prediction results, i.e., the forecasting, to be comprehensible and in various ways, e.g., as a chart or as a table. Moreover, they want an automatically calculation of the predictor performance (prediction error rates), which also should be displayed in various ways (chart or table).

The PredictionTool must provide the possibility to generate different clustering runs according to a given range of clusters number. The PredictionTool users desire an automatically calculation of the performance of each run according to the different performance criteria. Based on this evaluation results, the user can decide which clusters number must be selected for the prediction.

Functional Requirements

FR1.1 PredictionTool enables automatically forecasting of various project metrics for an ongoing project on the basis of the historical data. Based on this forecasting the user can estimate future project's state. The PredictionTool must provide an easy to use and self-explaining UI.

FR1.2 Straightforward design allowing good usability and configuration of the PredictionTool. The user should be able to obtain the desired results in an easy and quick way by performing only few actions.

FR1.3 The configuration of PredictionTool does not require expert knowledge in data mining. The Adjustment of clustering parameters, e.g., the maximum number of clusters in the end result, the maximum size of each cluster, etc., should be possible to fine-tune the forecasting.

FR1.4 PredictionTool must provide various import options to allow the import of the gathered historical data from different sources. Hence, the large data set is available for better prediction results.

FR1.4.1 The import function must support Excel format.

FR1.4.2 The import function must support SQL databases; e.g., MySQL.

FR1.5 Providing various regression and clustering methods. The PredictionTool must provide various data mining methods dependent on the input data to enable trying out various regression and clustering algorithms and identifying the *"best-pactictice"* methods combinations.

FR1.5.1 The function for calculation regression must support linear regression methods.

FR1.5.2 The function for calculation regression must support non-linear regression methods.

FR1.6 Providing evaluation components to enable estimation of prediction errors to measure the predictor performance.

FR1.6.1 The PredictionTool must support generation of multiple clustering runs for a given range of clusters number.

FR1.6.2 The PredictionTool must support different evaluation methods to estimate performance of each clustering run from a given range of clusters number.

FR1.6.3 The PredictionTool must support comparison of the evaluation results for all clustering runs from a given range of clusters number.

FR1.7 Generation of forecasting according to different project metrics. The system should enable prediction facilities independent from the used metric.

FR1.8 Providing configuration and visualization components to enable desired adjustments of PredictionTool as well as the analysis of prediction results based on different representation methods.

FR1.8.1 The function for presentation of calculation results must support tabular representation.

FR1.8.2 The function for presentation of calculation results must support graphical representation.

FR1.8.2.1 The function for presentation of calculation results must support different line charts.

FR1.8.2.2 The function for presentation of calculation results must support box and whisker diagrams.

3.2 Stakeholders and Requirements of the DataMiningFramework

By taking into account the tasks and goals of the underlying software system that were described in section 1.1, we identified the second group of stakeholders, namely, those of the DataMiningFramework. We derived the requirements for the DataMiningFramework based on the stakeholders' needs.

Stakeholders

The stakeholders of the DataMiningFramework use the framework for a tool in a domain which might be different from predicting project's development. Thus, they need to adapt the framework to a specific problem domain by extending or replacing the components of the system. The DataMiningFramework users are typically interested in an easy adaption and ease of use of the framework. In other words, this group of users is interested in the data mining capabilities of the system. Furthermore, the DataMiningFramework users might wish to integrate the DataMiningFramework into an already existing software system.

We identified different stakeholder roles for this stakeholder group: project developers, architects, and problem domain experts. According to this roles, the DataMiningFramework users have different needs. The project developers and architects desire to replace single components of the framework to adapt for the problem to be solved. All DataMiningFramework users expect that the framework provides the main steps for discovering data knowledge, i.e., data import, data pre-processing, data filtering, discovering, evaluation of data patterns, and visualization of obtained knowledge. The problem domain experts want to be able to apply different data mining algorithms. Moreover, they desire to be able to adjust the discovering data knowledge steps according to their needs.

Note that the DataMiningFramework users are not necessarily data mining experts as well. Nevertheless, they should be able to use the framework. According to the DataMiningFramework stakeholder and their needs we identified the following requirements to the DataMiningFramework.

Functional Requirements

FR2.1 Providing loosely-coupled architecture. Each component of the system has little or no knowledge about other components to obtain better modularity of the system

and independence of each component from the technology that is used for other components.

FR2.2 Each step of discovering data knowledge is encapsulated by a component of the system. Hence, each discovering data knowledge step can be implemented without considerations regarding the implementation details and the applied technology of the other steps.

FR2.3 Providing the necessary steps of discovering data knowledge as defined in section 2.3 to enable all tasks of data mining analysis.

FR2.4 Providing extendability and exchangeability of the system. The framework should offer the possibility to exchange or extend a component with little effort to support integration of the new data mining solutions.

FR2.5 Integrability of the system. The framework should keep open the possibility to be integrated into a measurement system. In this case, theDataMiningFramework serves as a module for different data mining tasks of a larger system.

Non-functional Requirements

Additionally, to the functional requirements, we defined non-functional requirements:

NFR1.1 The system should be a web-based application to enable platform independence.

NFR1.2 The architecture of the system should be divided into three layers: the client layer, the business layer and the data layer.

NFR1.3 The business layer of the system should be implemented with the EJB framework, described in section 6.1.1.

NFR1.4 For realization of the data layer a MySQL database should be used

NFR1.5 The GlassFish application server has to be used for development and deployment of the PredictionTool.

4 Related Work

Contents

4.1 Software Project Schedule Prediction Using Bayesian Network 25
4.2 Prediction of Defects in Software Projects 26
4.2.1 Theoretical Framework for Defect Prediction 27
4.2.2 Prediction of Defects by Applying Clustering Approach 28
4.2.3 Defect Prediction based on Retrospective Project Analysis 29
4.2.4 Defect Prediction by Using Classification 30
4.2.5 Defect Prediction by Using Support Vector Machines 31
4.2.6 Defect Prediction by Using Belief Networks 31
4.2.7 Prediction of Defects by Using Neural Networks 32
4.2.8 The Cross-project Defect Prediction Model 33
4.2.9 Defect Prediction by Using Method-Level Metrics 35
4.3 Discussion 36

Increased complexity of software development requires estimation and prediction of the software projects effort as early as possible. A reliable forecast of software project progress improves cost control and accuracy of quality. Due to the grown interest in this topic in academia, there exist a number of approaches investigating the issue of prediction.

In this chapter, we present an overview of different research papers on this topic, approaches, and related problem domains. Further, we discuss the benefits and disadvantages of proposed methods and contrast these to the requirements of this thesis. Finally, we summarize the introduced approaches and discuss possible application and enhancements with respect to our work.

4.1 Software Project Schedule Prediction Using Bayesian Network

Wang et al. present an approach based on a Bayesian network to predict software project schedule [WWM10]. A Bayesian network is a graphical probabilistic model of causal relationships which provides structure and parameter learning. The model is defined as a directed acyclic graph, where the nodes represent a set of variables and edges represent a conditional dependence of the corresponding variables [HK00].

In their work, Wang et al. use a Bayesian network to model uncertainty of project schedule. The nodes of the model represent influencing factors of a project schedule. The proposed method includes three steps. First, the model data is collected and influence

factors of each phase of the software development process are identified. Analysis of these factors is used to define cause-effect relationships between nodes. The collected data needs to satisfy such characteristics as truth, synchronization, consistency, and validity to ensure the accuracy of the Bayesian model. In reality, the values of raw data may vary from the range of the required feature values listed above. Hence, in the second phase, the data is processed. In this phase, limits for acceptable variance in data are established. Additionally, Wang et al. examine the cause of the data variance. Afterwards, the data is dynamically discretized by transforming the data values into the interval $[0, 1]$. In the third and the last phase, two learning algorithms are applied: the structure learning approach and the parameter learning method.

The proposed approach might be used by project managers when planning a software project. A project manager might be interested in how the current project schedule varies from the planned project schedule, i.e., they want to know the project schedule variance. To analyze the approach's results, the authors concentrate on several aspects of the problem domain. Specifically, the calculation of the project schedule variance from a known node of a project schedule, applying this information to re-plan a project to finish it in time and identifying variables that are important for the software schedule variance.

The evaluation results show the validity of the proposed approach. To this end, Wang et al. applied the approach in two projects. The predicted project schedule implied that the project duration will exceed the original estimation by less than 10% for the first project, and between 10% and 20% for the second project. The real values of the first and second project's delay are 7.4% and 12.5%, respectively, which proved validity of the proposed model for both of the experiment's projects. Furthermore, the authors performed a sensitivity analysis to discover the influence factors on software schedule. The identified factors help to decide which node of the Bayesian model should be adjusted to reduce schedule variance.

In addition, the approach offers an opportunity to identify and filter the factors that influence the process quality. However, the authors check the validity of the approach by analyzing only two projects. They also provide a little information about the effectiveness of the approach with respect to the tools to be used and the metrics to be applied.

4.2 Prediction of Defects in Software Projects

The approaches for prediction faults in software is a large area of interest today. The ability to predict the defects in software projects helps to achieve effective development and management processes. As a consequence, a large number of software defects prediction models can be found in literature. Since the underlying problem domain is collateral to the endeavor of this thesis, we will present several approaches to software defects prediction and discuss their results, subsequently.

4.2.1 Theoretical Framework for Defect Prediction

In [WRB08], Wahyudin et al. describe a theoretical framework for software defect prediction. The authors summarize a number of published studies on defect prediction and derive common requirements in their work. As a result, a generic framework is given. Additionally, the authors perform a paper review and analysis in order to evaluate the framework.

The presented framework for software defect prediction includes three phases. The first phase is called the preparation phase and consists of three steps, where the prediction goal-relevant questions and hypotheses are specified, and the feasibility of the initial goals of the prediction is determined. In the second phase, the data for model training is collected. Then, the prediction model is trained by applying standard statistical or machine learning tools, and validated to assess its performance. In the last phase, the model is used to perform defect prediction and the robustness of the model is analyzed.

Wahyudin et al. carry out a comprehensive literature review to find information relevant to the research questions and the search process. The observed studies are summarized according to the framework's phases and essential steps. As a result, the authors report that many studies do not explicitly describe the goals and the target stakeholders of the study. They state that the goal definition and hypotheses formulation in the first phase of the presented framework are essential for defect prediction with different levels of detail and presentation. According to the literature review for the second phase, Wahyudin et al. conclude that most studies use Spearman bivariate correlation analysis and linear regression for variable selection. The authors suggest that the second phase of defect prediction model should be extended with internal validation methods and model performance measures like they have done in their study. Regarding the third phase, the authors report that most observed studies do not provide external validation and robustness analysis of the suggested prediction model. In most cases, the reason is low performance of their prediction model.

Concluding, several implications can be derived as stated by Wahyudin et al. First, there exist many approaches that do not provide reliable results in defect prediction due to missing or incomplete information. Thus, the data has to be collected and analyzed very carefully. Second, in order to provide trusted results, an early and quick feasibility estimation of a defect prediction method with acceptable performance is required. Last, additional quality validation is needed, i.e., a measure to verify the obtained prediction results. The authors state that the indicated issues are crucial when applying defect prediction in practice.

Furthermore, Ramler et al. report experiences and empirical results obtained from applying the defect prediction framework in [RWS+09]. Furthermore, the authors discuss the essential issues related to the application of defect prediction models in practice. As mentioned above, the authors examine defect prediction for use in large software development projects for planning the system tests. For this purpose, the defect prediction framework developed in [WRB08] is applied. The prediction model is based on historical data which is used in a learning algorithm. The authors state that the effectiveness of a prediction model depends on the prediction objectives. Therefore they

perform an analysis and observe several questions in order to identify key decisions while constructing the defect prediction model. In particular, what learning algorithm should be used and how can the performance of the model be measured.

As a result of the analysis, Ramler et al. determine that for each project version a prediction model should be constructed using the data from the previous version. Such an approach has the potential to enable a prediction with high precision. The authors state that good performance can be obtained by applying a learning algorithm, which captures the current state and repeats what is already known. With respect to the choice of a learning algorithm, the authors state that it should satisfy certain criteria like prediction performance, ease of use and comprehensibility of the prediction model. A significant issue of the performance of learning algorithms is the ease of tuning the algorithm's parameters. The authors suggest to apply the learning algorithms implemented by the tools WEKA and MLF, as the both show a good applicability in practice. Last but not least, the authors suggest to use a confusion matrix in order to quantify the prediction performance.

During the experiment, seven versions of a software system were analyzed. The prediction results were then compared with the real values, where the highest accuracy was achieved for version 4 with about 78% and the lowest accuracy was achieved for version 1 with 67%. Ramler et al. state that the prediction framework shows good results with respect to applicability and practical performance. However, a better learning algorithm could increase the prediction performance.

4.2.2 Prediction of Defects by Applying Clustering Approach

In [JM10], Jureczko and Madeyski describe a clustering approach of software projects, where groups of software projects with similar characteristics regarding defect prediction are identified. In other words, a defect prediction model can be applied to all projects which belong to the same group (or cluster). The authors use hierarchical and k-Means clustering, as well as Kohonen's neural network for building clusters with similar elements. After identifying clusters, a statistical analysis is applied in order to verify the obtained groups. For each cluster, a defect prediction model is created using suitable metrics and stepwise linear regression. In particular, there are two prediction models created. One model deals with projects that belong to a given group, while the other is applied to all remaining projects outside of any group.

Jureczko and Madeyski performed the evaluation by using various metrics. Besides the lines of code metric, the authors use several other metrics, e.g., weighted methods per class, depth of inheritance tree, coupling between object classes, lack of cohesion in methods, number of public methods, data access metric, etc. After applying the approach, six clusters based on correlation vectors were identified, where the existence of two clusters was proven. The authors compared the obtained result with other studies and determined that there are no significant inconsistencies. However, the authors state that the proposed approach can be difficult to reproduce in the industry since information about defects is necessary for constructing the correlation vectors. The proposed approach needs for further investigation in order to be applied in practice.

4.2.3 Defect Prediction based on Retrospective Project Analysis

In [HGW11], Herbold et al. present a retrospective project analysis using the Expectation-Maximization (EM) clustering algorithm. The goal of the proposed research work is to develop a project prediction approach that allows to map the clusters to phases of the project. The analysis is data-driven and is based on software metrics measured during different phases of the project. The authors use lines of code (LOC), number of bugs (BUG), and number of active bugs (ACTBUG) as the main metrics. Here, BUG measures the total number of bugs that are not fixed, while ACTBUG identifies the currently know bugs.

The EM algorithm requires data collection and data preparation during the first two steps. The collected data should be normalized to reduce the impact of different metric scales. In the third and the last step of the algorithm, data analysis and estimation of the data sources, e.g., the project phases, is performed.

In order to validate the approach, Herbold et al. performed a study with two main questions: whether the presented approach allows to identify project phases, and whether either the BUG metric or the ACTBUG metric is sufficient, or both metrics are required. For their study, the authors applied their approach to two large-scale open-source projects and use Weka's implementation of the EM clustering with a maximum of 100 iterations and a dynamic number of clusters.

After analyzing the experiment's results, the authors state that the approach is indeed able to split a project into phases; however, this worked only for one of two projects. In fact, the last phase of the project can be identified with high accuracy. For the other phases but the final phase, the experiments show that the approach is unreliable. Furthermore, the authors report that for the other project the approach yielded no detected phases at all. Regarding the question addressing the metrics, the authors determined that there is no significant difference between applying only one metric, BUG or ACTBUG, and applying both. Thus, only one of the metrics is sufficient to optimize the approach.

In summary, the evaluation results show that the approach provides reasonable prediction accuracy only for the last phase and only for one of two tested projects. Note that the authors applied their approach only to two projects, which makes statements regarding the results of the presented approach unreliable. As a consequence, the proposed approach cannot be considered practicable and needs further research.

In [HGNW10], Herbold et al. present a similar approach, which uses k-Means clustering for the retrospective analysis of software projects. The analysis focuses on separating the data into two sets. The first set includes the versions before the feature freeze and the second set the versions after the freeze. As the approach is based on k-Means clustering, k clusters are identified, represented by their centroids.

The authors validated their approach by applying it to two projects, where they use tuples of metrics (LOC, BUG). As a result, the software versions were correctly clustered. However, the authors state that their approach needs to be analyzed with more confidence and plan to apply the approach to more projects using additional software metrics. Therefore, it is hard to tell whether the approach can be successfully applied

in practice.

Ramler and Himmelbauer consider in [RH13] noise in bug report data and its impact on defect prediction. To this end, the authors evaluated predictions for different versions of a large industrial software system. The approach predicts data of version $n+1$ based on the data of version n. The data set of noise vectors is evaluated in version $n+1$ in order to indicate prediction accuracy w.r.t. a certain chance level. The authors assume that an experiment instance can be considered successful, if the prediction accuracy is above of chance level of at least 95% of the predictions. The results of the first part of the experiment show that the average prediction accuracy is significantly higher than the average chance level. The experiment indicated only a few cases where the prediction accuracy remains below the chance level. In the second part of the experiment, a model of version n is used for predicting version $n+1$ by different noise levels. The results demonstrate that the prediction model is reliable and remains above 95% at a noise level of up to 20%. Thus, the approach seems to be useful and can be applied in practice.

4.2.4 Defect Prediction by Using Classification

Ma et al. present a comprehensive study of two algorithms for the associative classification approach for software fault prediction in [MZC+14]. The classification is based on a subset of association rules. The approach includes four main stages: data preprocessing, rule generation, classifier building and prediction, as well as prediction results evaluation. During the rule generation phase, optimal minsup and minconf thresholds are determined. These parameters are used to decide whether a rule is accepted or rejected. Using obtained thresholds, the class association rules (CARs) are generated. The size of the generated rule set is controlled by using several rule pruning strategies, which apply pessimistic error rate pruning. After the CARs have been generated, the final classifier is constructed to enable subsequent inference on the test data, where the confidence of each rule is used to estimate the prediction probability.

During the experiment, the authors compared the proposed associative classification approach (AC) with five different tree-based and rule-based classifiers: C4.5, Decision Table (DT), CART, ADT, and RIPPER. The discussion of the experimental results indicates that among the different classifiers DT and CART outperform the other classifiers w.r.t. accuracy and specificity. However, AC and RIPPER show the best results when considering geometric mean and F-measure, a harmonic mean of recall and precision. The authors state that AC and DT perform better over the methods on the area under the ROC curve (AUC), which is the most informative indicator of predictive accuracy.

Regarding the results of cross-project validation, the authors report that classifiers trained on four data sets achieve an acceptable prediction performance with relative AUC values above 90%. The AC classifier shows a high capability of generalization on the data set with a high mean value and low standard deviation.

4.2.5 Defect Prediction by Using Support Vector Machines

In [EE08], Karim et al. describe an approach for software defect prediction by using support vector machines (SVM). The support vector machines approach is a kernel-based learning algorithm that allows for minimization of the generalization error. The SVM classifier uses a number of support vectors to define the boundary between two-class pattern recognition problems. According to the authors, the main characteristics of the SVM approach are that the SVM classifier is independent from the feature space dimensionality, it gives a global optimum solution, it is robust to outliers and it can model non-linear functional relationships. In their paper, the authors provide the mathematical background and explanation of the SVM algorithm.

In their study, Karim et al. evaluate the SVM approach by comparing it with eight well-known statistical and machine learning models. In particular, they compared the SVM approach with logistic regression (LR), k-nearest neighbor (KNN), multi-layer perceptrons (MLP), radial basis function (RBF), bayesian belief network (BBN), naive Bayes (NB), random forest (RF), and decision tree (DT). For this purpose, they use four data sets (CM1, PC1, KC1, and KC3) obtained from NASA software projects, where each data set includes 21 software metrics.

According to the results obtained from the CM1 data set, the SVM method provides better outcomes then the other eight models with respect to accuracy (90.69%) and a perfect recall (100%). Furthermore, the outcomes of six of the eight compared models in precision (90.66%) except for BBN (94.17%) and NB are shown.

The results from the PC1 data set show that RF outperforms the other methods with respect to accuracy by 93.66% and F-measure by 96.7%, which is not significantly higher than the accuracy (93.10%) and F-measure (96.4%) achieved by the SVM method. The KNN approach shows the highest precision (95.54%) in this case. The SVM perform better than the other methods by recall with 99.47%.

The results of the experiment obtained from KC1 and KC3 data sets reflect that SVM achieves the highest recall for the both data sets. However, with respect to accuracy, precision and F-measure the SVM algorithm is outperformed by MLP, BBN, RBF, and NB.

The authors summarize that the proposed approach is at least competitively viable against the compared algorithms for predicting defect-prone software modules, since it outperforms many models in recall and F-measure. However, the approach shows worse results than the other methods in precision and accuracy, which is a motivation to carry out further empirical studies with more data sets to improve the proposed method.

4.2.6 Defect Prediction by Using Belief Networks

The approach for prediction of fault prone modules described by Guo et al. in [GCS03], is based on Dempster-Shafer belief networks. The approach uses a directed graph whose nodes are connected by implication rules. During the experiment, the authors consider whether the module contains any defects, instead of the number of defect modules. They identify 21 metrics in the given data sets. These metrics are called predictors. They

are chosen by a logistic procedure which leads to different sequences of the predictors. The predictor sequences are analyzed where the best one is noted. Additionally, there are other tuning parameters in the Dempster-Shafer networks. The approach allows for selecting the optimal results for each set of criteria.

For the evaluation Guo et al. used two binary data sets that were generated by partitioning with mean and median values. They compare the Dempster-Shafer approach with discriminant analysis, logistic regression, and ROCKY. According to the evaluation results, the suggested approach outperforms the logistic regression and discriminant analysis methods in accuracy. After comparing the presented approach with the ROCKY defect prediction method, the authors state that predictions done by ROCKY lead to more effort than D-S network predictions. The authors inspected about 72% of the code (LOC) by using the D-S network model which showing that 91% of the modules are fault prone. By applying the ROCKY approach, it was necessary to read 94% of code to discover the 91% fault prone modules. Thus, the approach seems to be useful for real-world applications in order to predict software quality. Further, it provides various optimization criteria and allows for integration of new optimization requirements.

4.2.7 Prediction of Defects by Using Neural Networks

Mahaweerawat et al. describe an approach to predict faults in object-oriented software by using neural networks in [MSLM04]. Besides fault prediction, the approach provides a method for the classification of fault type. In particular, the authors consider faults due to inheritance and polymorphism. They assumed that inheritance and polymorphism might be significant factors that cause anomalies and faults of software products.

The approach is based on two neural network techniques: Multi-layer Perceptron (MLP) and Radial-basis Function Networks (RBFN). While MLP provides clustering of the input data according to appropriate fault categories, RBFN helps to indicate fault type by using curve-fitting approximation. To this end, the authors define the prediction of fault type as a clustering problem where a cluster represents a fault type.

For their study, Mahaweerawat et al. initially applied a set of 60 object-oriented metrics. In order to filter only relevant metrics, the authors propose an algorithm which allows to select appropriate metrics. The algorithm allows to set an initial weight of each metric. In the next step, pairs of fault-free and faulty classes from the training set are established and the values of differences between this pairs are calculated. The authors state that the mechanism of calculating differences prevents metrics intermix. The differences are calculated in several iterations and used to adjust the weight value of each metric. The algorithm terminates when all fault-free classes match with all faulty classes. In the last step, the metrics with weight values above the selected threshold are selected.

In order to evaluate the approach, the authors used three data sets to which both neural network techniques, MLP and RBFN, are applied. For this purpose, the authors used the metrics which are filtered with the algorithm described above. The aim of the experiment in case of MLP is correct classification of the data points into fault-free and faulty classes. RBF neural network is used to categorize the faults with respect to

defined fault types.

Mahaweerawat et al. report that the two predictive models allows to predict faultiness of a class with more than 90% accuracy. The results of the evaluation show that the faulty classes can be detected in 91.53% of test cases. The inspection cost for verification is 59.55%, and the waste cost is 3.51%, and only 2.09% of faulty classes are undetected. The faulty type prediction results in 90.05% of test cases. The authors assume that the reason of misclassification might be the reason of overlapping parameters of fault types.

The results obtained from the experiment show the potential of the proposed approach in the software fault prediction domain. Additionally, the authors developed an algorithm which allows to filter for relevant metrics. However, the application of neural networks requires a huge set of data, and thus, the analysis of such amount of data might be very costly. Furthermore, the authors mention that the proposed approach can be enhanced by adding more parameters and proper data classification technique.

In [QT03], Quah and Thwin describe two kinds of defect prediction of object-oriented software quality using neural network models. Specifically, the authors analyze predicting the number of defects in a class and predicting the number of lines changed per class. For this purpose, two neural network models are applied: the Ward neural network and the General Regression neural network (GRNN). The Ward neural network is a back-propagation network with various activation functions, where a Gaussian function is applied in order to detect values in the mid-range of the data and a Gaussian complement is used to indicate features for the upper and lower extremes of the data. The General Regression neural network is a memory-based network that provide estimates of continuous variables and converges to the linear or nonlinear regression surface. The algorithm is a one-pass learning algorithm, which allows quickly training of instances.

Quah and Thwin use object-oriented metric approach and traditional complexity metric approach as a measurement method, where the metrics are used as independent variables. The performance of the prediction algorithm is determined by applying the coefficient of multiple determination (R-square), the correlation coefficient, r-square, mean square error, mean absolute error, minimum absolute error and maximum absolute error. The authors state that the presented approach shows a high degree of confidence for successful validations with both the Ward neural network and the General Regression neural network, where the GRNN network model is more accurate than the Ward network model.

4.2.8 The Cross-project Defect Prediction Model

In [ZNG+09], Zimmerman et al. suggest the cross-project defect prediction model, where possible defects of a project can be predicted by taking prediction results from another project. The authors consider two dimensions of cross-project defect prediction: the domain dimension and the company dimension. In order to evaluate the validity of the approach, the Firefox and Internet Explorer web browsers are used to predict software defects for each other. The software choice is of the 'same domain, but different companies' category.

The metrics used to perform the experiment are: the number of observations (file

count, binary count, component count), the total lines of code (LOC), added LOC, deleted LOC, modified LOC, and the number of edits (commits), cyclomatic complexity, number of bugs (pre-release), number of developers, number of defects (post-release). The cross-project prediction works only if the two projects (A, B) are chosen in a compatible way: the model of project A predicts project B, if the products of A and B are different or B has a later version than A.

The measured data is classified into non-defect-prone and defect-prone by applying a statistical low confidence bound and a logistic regression model. Metrics with fewer defects than the low confidence bound are defined as non-defect-prone, otherwise the metric is classified as defect-prone. Zimmerman et al. evaluate precision, accuracy, and recall to determine how well the data is classified as defect-prone in B according to the model from A. The authors state that for some project combinations, the logistic regression model provides the same prediction for all elements of the project, which is unusable in practice.

After analyzing the experimental results, Zimmerman et al. found that a project can be considered a strong predictor for another project only if precision, accuracy, and recall are greater than 0.75. According to the experiment's outcome only 3.4% of the cross-project combinations satisfied this criteria. Further, application of the approach on projects in the same domain does not provide a proper prediction model. The data needs to be analyzed and processed before a prediction model is build. Thus, the approach is only able to correctly predict faults in few projects, which shows that it cannot be applied in practice.

Another approach of cross-project prediction is described by Canfora et al. in [CLP+13]. The proposed model is based on the results obtained by Zimmerman et al. outlined above. Canfora et al. note that the success of cross-project prediction depends on the heterogeneity of the project domain, source code characteristics, and process organization. To minimize error in the prediction model, the authors offer a multi-objective approach that suppose to offer a better cross-project prediction. Further, the approach allows to choose predictors in order to identify a high number of defect-prone artifacts. For this purpose, the authors applied a multi-objective genetic algorithm, which uses the product metrics as independent variables to build a logistic regression predictor.

After applying of the approach to perform cross-project prediction with 10 software projects, Canfora et al. compared multi-objective cross-project prediction with single cross-project prediction and within-project prediction. The results show that within-project prediction outperforms the cross-project prediction w.r.t. precision and recall. However, the authors state that the proposed approach allows to generate information about how much code needs to be analyzed to achieve a certain level of recall. In addition, the authors tried to achieve a tradeoff between precision and inspection costs, since a low precision implies a high code inspection cost.

According to the study's results, the proposed multi-object cross-project prediction model offers better cost-effectiveness and higher precision than a single-objective predictor does. Furthermore, the approach allows to choose a configuration for the

predictor that best fits the specific situation. However, the proposed approach does not show significant difference to the approach presented by Zimmerman et al. w.r.t. precision and recall. Furthermore, the authors mention that in general the cross-project prediction method provides distinctly worse results than within-project prediction and needs more investigations. Hence, the presented approach is not the best solution for practical application.

4.2.9 Defect Prediction by Using Method-Level Metrics

In [CD13], Catal and Diri describe a fault prediction strategy based on method-level metrics (e.g., lines of code, cyclomatic complexity, essential complexity, unique operator, etc.) with constant threshold as a filtering mechanism. The approach allows for software-based fault prediction without information about previous fault data. Furthermore, the authors emphasize that the proposed approach offers a fully automated prediction method, meaning no expert's review is needed; e.g., to label obtained clusters. There are several benefits to such a human-expert independent approach. For instance, the non-existing difficulty and additional effort of embedding an expert into a software tool.

The prediction strategy of the presented approach is divided into five steps. In the first step, the best metric set fit to the fault-proneness measurement is identified. Catal and Diri used the experiences of the previous related studies to select the metrics. A clustering-based fault prediction requires a human expert to label the clusters after using clustering methods. As mentioned above, the authors suggest an automated mechanism to label software modules as fault-prone or non-fault-prone. For this reason, the metric set obtained from the previous step is used to select appropriate metrics for the labeling of a software module. In the next step, the most appropriate filtering mechanism is selected, where a module falls under a fault-prone. For this purpose the authors apply the marginal absolute semantical filter with the specified threshold value, where modules are separated into a fault-prone or a not fault-prone group. In the fourth step, the metric thresholds are defined. The fifth and the last step deals with the merging procedure. Using a composition mechanism, where an analyzed module is labeled as defect-prone if at least one metric has a value which is equal or greater than the threshold.

For the evaluation. Catal and Diri used error, false positive rate (FPR), and false negative rate (FNP) as performance indicators. Here, the error parameter reflects the amount of mislabeled modules, FRP defines the number of non-faulty modules labeled as fault-prone, and FNP the faulty modules labeled as non-fault-prone. The authors then compared their approach with a related study based on a clustering method and a software expert's review. The authors claim that their approach shows better results w.r.t. the FNR values. However, error and FNP rates are higher than in the compared study. The authors assume that the reason lies in data noise and a large number of outliers. In order to obtain better results, the authors suggest to apply a threshold calculation approach instead of a constant method.

The presented approach shows an attempt to develop a fully automated prediction model, i.e., it does not require a data mining or software engineering expert. However,

the evaluation results show high FNR and high overall classification error rates. For instance, the percentage of non-detected faulty modules lies over 30% which can cause dramatic failures in the software. Therefore, the presented method requires more research and enhancements before being applied in the industry.

4.3 Discussion

Concluding the literature review, we can state that most approaches do not provide reliable results for large test candidates sets. Further, many approaches provide a little information about the used tools and the applied metrics. However, the main issue is that most methods use static metrics such as lines of code, number of bugs, weighted methods per class, depth of inheritance tree, etc. for the prediction. Hence, the proposed approaches are impractical for dynamic metrics (e.g., CPI and SPI) predicting project behavior.

The theoretical framework for software defect prediction described by Ramler et al. can be used as a starting point for our approach. Their framework implements the main steps of the knowledge discovering process: data preparation, prediction model construction, and analysis for prediction model robustness. We adopt the concept of this theoretical framework to our needs and problem domain. To this end, we take into account pros and cons of the analyzed literature, and with that, extend the framework by additional phases (and steps within phases) in order to obtain better prediction results.

5 Concept

Contents

5.1 Idea of the Framework 37
5.2 Processes Modeling 38
5.2.1 Data Import and Pre-Processing 40
5.2.2 Classifier Construction and Evaluation 44
5.2.3 Prediction 45
5.3 GUI Prototyping 46
5.4 Domain Model 50

According to the objectives defined in chapter 1 and requirements identified in chapter 3, in this chapter, we design a framework for automatic prediction in different application domains. Additionally, we design a prototype for a tool-based forecasting of project's development as decision making aid for project managers. The resulting forecasting tool enables creating a project-specific classifier based on historical data and represents the *prove of concept* of our framework.

In this chapter, we present and analyze the conceptual issue of our research work. First, in section 5.1, we present the abstract idea of our approach. After that, we present and explain the framework activities and processes on the basis of flow diagrams in section 5.2. Third, we depict the GUI prototypes of the tool in section 5.3. Finally, we present our domain model for the framework in section 5.4.

5.1 Idea of the Framework

Before we can design the architecture of the framework, we have to define the basic concept of the future software. The concept definition is a phase of software development that is established between the requirements engineering and architecture design phases. The endeavor of this phase is domain abstraction, which is needed to identify the concrete components of the software architecture. In this section, we introduce the idea of the problem solution and explain the essential parts of the DataMiningFramework.

The main idea of the DataMiningFramework is based on identifying similarity patterns in the data, which are, then, used to construct a classifier. With this classifier, we classify the ongoing project and derive predictions. To do so, we follow the three-phases breakdown of Ramler et al.'s theoretic framework: *preparation*, *classifier construction* as model creation, and *prediction* as model usage (cf. [WRB08]). Clearly, the steps within those phases differ from Ramler et al.'s theoretic framework, however, are consonant to the five steps of the knowledge discovering approach as described

in section **??**: First, *data import* and *pre-processing* in the preparation phase. Then, *clustering* and *labeling* in classifier construction. Last, *predicting project values* in prediction.

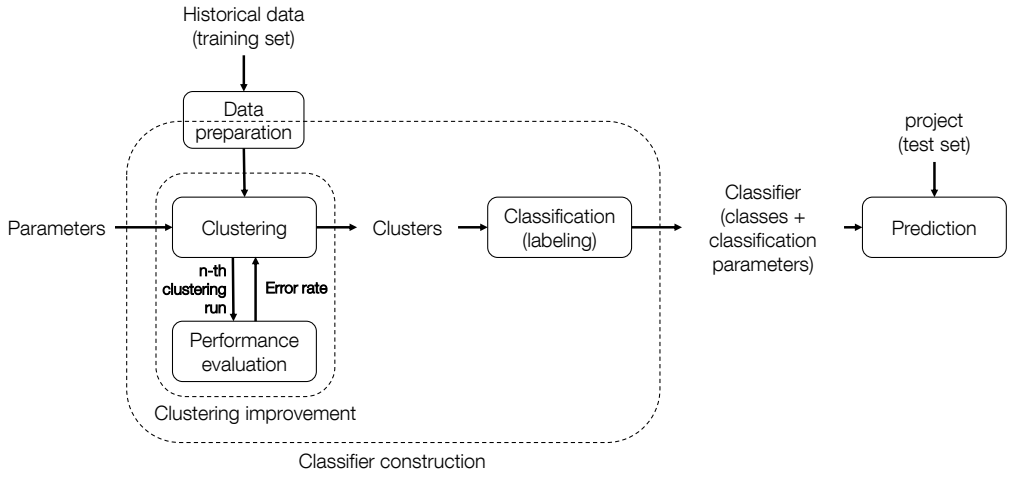

Figure 5.1: Abstract control flow of the framework

Figure 5.1 depicts the abstract control flow of our approach. First, we apply data mining methods such as clustering to partition the historical data, i.e., the training set, into clusters according to a certain similarity principals. Using performance evaluation an optimal number of clusters is identified. Once the clusters are computed, the identified groups of similar projects are *labeled* by the user or an external system. With this, we obtain a classifier, which comprises the labeled clusters, i.e., classes, and classification parameters. Finally, we use this classifier along with the ongoing project, i.e., the test set, to make the prediction.

Often, the input for the classifier construction, i.e., the historical data, needs to be prepared beforehand. For instance, the given time series of past projects are of different durations and might include outliers; both negatively affect the clustering result. Hence, we need to prepare the data in order to perform a meaningful clustering; e.g., cleaning data by removing outliers, performing data normalization, and computing missing values. In the following sections, we will describe the phases and steps of our approach in more detail.

5.2 Processes Modeling

In this section, we introduce the process model that reflects the essential framework activities within each step. For this purpose, we use flow diagrams, due to their simple

notation as illustrated in figure 5.2.

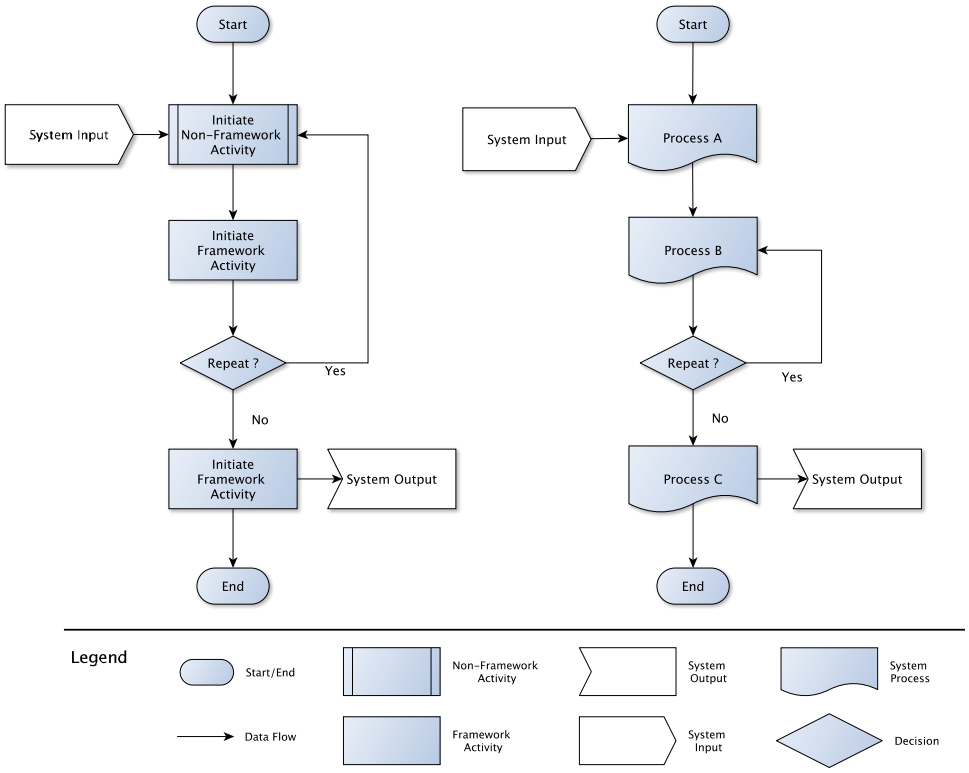

Figure 5.2: Example: Flow diagram elements

The start and end states in a flow diagram have a form of the rounded rectangles. Solid arrows describe the control flow of the process. The elements between start and end states describe either a process or a sub-process within the system. We distinguish between activities, denoted by rectangles, and system processes, denoted by rectangles with curved bottom. The former can be either framework (regular rectangles) or non-framework (rectangles with two vertical bars), i.e., carried out by the user or an external system, activities. The latter reflect a group of activities or processes. Diamond-shaped element refer to decisions, where conditions of decision are denoted by labels off the outgoing arrows. The flow continuous only by the arrow, whose condition is satisfied. The system's input and output are illustrated in flowcharts as the convex and concave polygons, respectively.

Figure 5.3 shows the overall procedure of the system described in section 5.1. To start with, the user initiate data import, which triggers the pre-processing step on import completion. Once the pre-processing is done, the system starts classifier construction

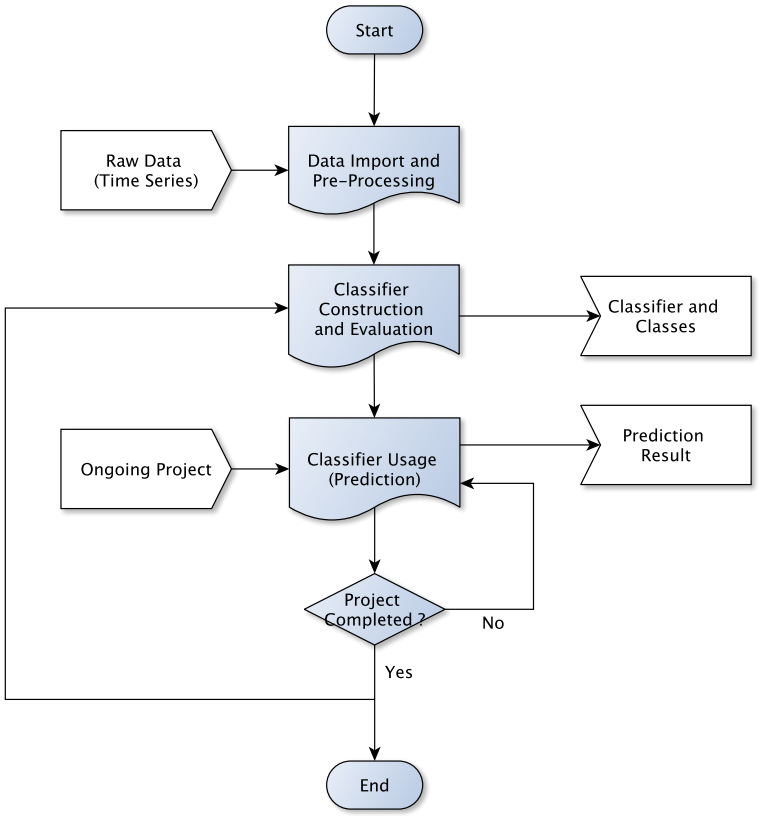

Figure 5.3: Overview of the overall approach

including clustering, classification, and evaluation of classifier. The output of the classifier construction process consists of the classifier along with the classes. Last but not least, the classifier is used to classify the ongoing project and results in a prediction for future values. We keep predicting the project iteratively during its development. Thereby, the project might change its class, and consequently, its predicted values are recalculated. After the project is finished, it will become a part of the training set in order to be used to construct classifier for the future projects. Subsequently, we describe each process of the system in detail.

5.2.1 Data Import and Pre-Processing

In this section, we describe the data import and data pre-processing process, which is depicted in figure 5.4). With respect to requirements **FR140, FR141, FR142**, the system must offer various possibilities of data import. The flowchart in figure 5.4 illustrates three data import options: file, database, or URL. The import is initiated either by the user or an external system. We require that the imported data are time series.

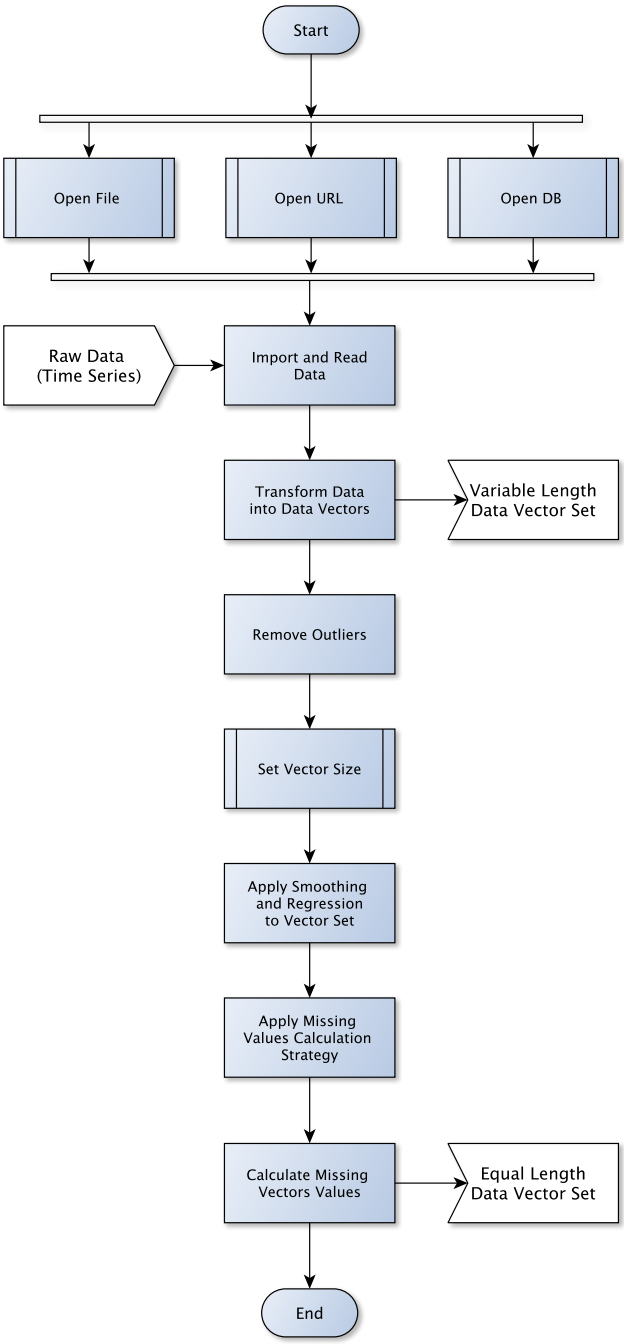

Figure 5.4: Overview of the import and pre-processing processes

After the import of the raw data, the framework starts transforming the data into the form that is appropriate for a meaningful clustering. Thereby, the system converts time

series into *data vectors*, which are needed for further processing. In the next step, the system must clean the vector set from the outlier for later clustering analysis. Thereby, the system removes all vectors which contain values exceeding a predefined threshold. After that, the user or an external system is engaged to meet decision regarding size of vectors, which means a number of values per vector instance. As explained above, the clustering tool expects that each instance to be clustered has the same number of values. To this end, the system must provide different strategies to perform normalization of data vectors with different lengths. In context of this work, this strategies are based on regression methods described in section 2.3.2.

In the data mining literature, regression analysis is often used for numeric prediction. However, in this thesis, a numeric regression cannot be applied for forecasting future project values. Instead, regression analysis is used in the data pre-processing step to estimate missing values. When the regression function is computed, it is applied to either stretch a vector by adding additional value, or to fill the gaps inside vectors by inserting estimated values between two given neighboring values.

At the end of the pre-processing process, the system outputs the set of normalized data vectors, i.e., vectors with equal length. This output serve as input for the next process, classifier construction.

Optimization Step

Since even after the data normalization, outlier might significantly affect clustering results, we apply *data smoothing* algorithms to remove noises from the data, and thus, achieve better regression results. To this end, values of time series are modified with intention either to reduce the values that are higher than their immediate neighbors, or to increase the values that are lower than their immediate neighbors. Han and Kamber describe several smoothing methods, which can be applied to time series [HK00].

In this work, we use a simplest smoothing algorithm based on the *moving average* method. The method replaces each data value with the average m of adjacent values, where m is a positive integer called the *smooth width* [O'H08]. In context of this work, we use a smooth width of 3, i.e., $m = 3$, and calculate smoothed values as follows:

$$s_i = \frac{x_{i-1}+x_i+x_{i+1}}{3},$$

where s_i is the smoothed value of the origin value x_i. In contrast to the original algorithm, we also smooth the first (x_0) and last (x_n) points with

$$s_0 = \frac{x_0+x_{i+1}}{2} \text{ and } s_n = \frac{x_{n-1}+x_n}{2}.$$

Figure 5.5 demonstrates an example of time series smoothing. The blue line with circles represents the original data, while the orange line with squares depicts the smoothed data values by applying the smoothing method as explained above. Figure 5.6 illustrates the resulting regression lines for the original (blue dashed line) and smoothed data (orange solid line), respectively. Here, we can see that after smoothing, outliers within the time series have less influence on the regression.

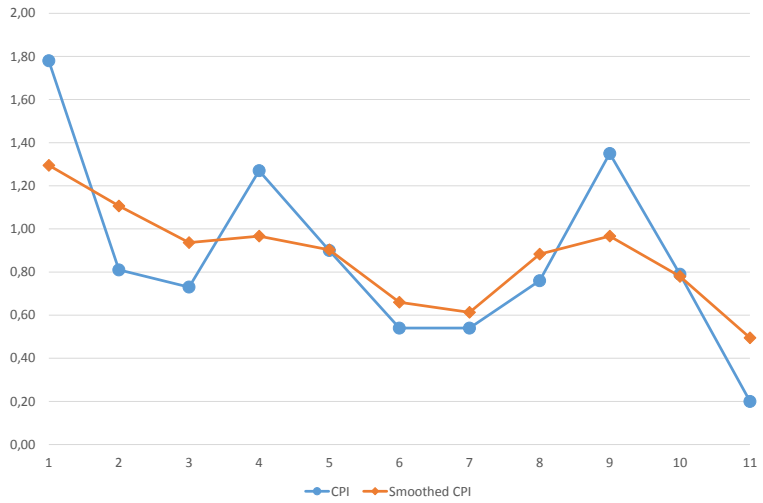

Figure 5.5: Example: time series smoothing

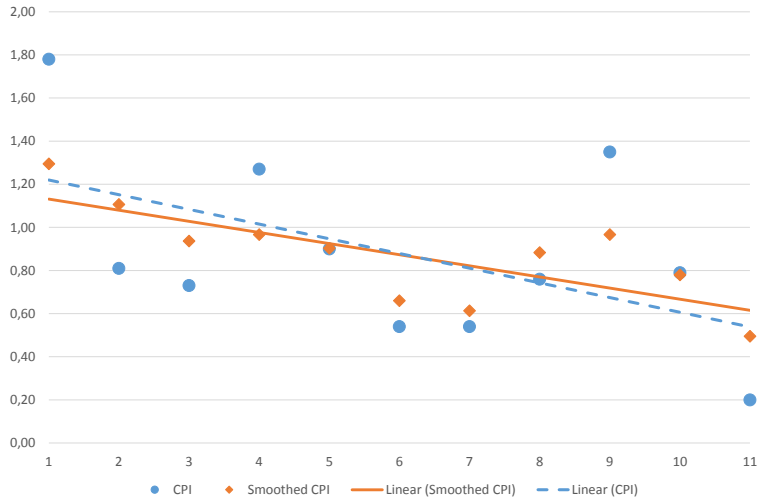

Figure 5.6: Example: time series smoothing with regression line

5.2.2 Classifier Construction and Evaluation

The construction of the classification model or so-called classifier starts with the adjustment of clustering setup parameters (cf. figure 5.7). That is the question of the conception task, which setup combinations should be selected. Furthermore, during designing the framework, we had to decide, which parameter settings must be implement as non-framework activity, i.e., the clustering parameter that the user should adjusts or another system before trigger a clustering procedure. If the user is not familiar with the data mining area, this task might be quit difficult for them. On the other hand, letting the framework decide what clustering parameters should be specified, restricts user's activities and the system application for the solution of a particular problem.

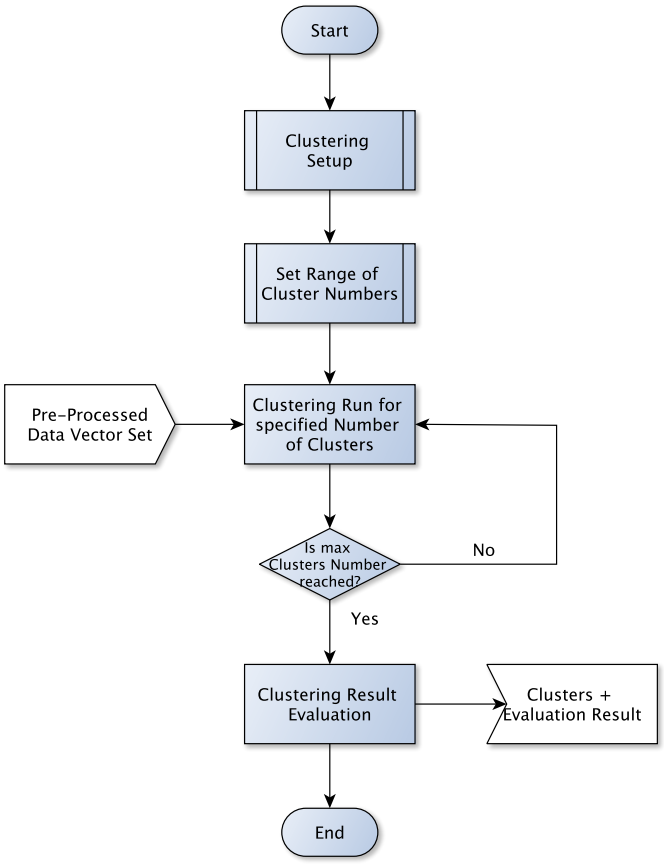

Figure 5.7: Overview of the classifier construction process: clustering

Thus, we resolve this issue by asking the user to specify the clustering algorithm used, the maximum cluster size (the maximum number of instances inside each cluster), as well as the range of cluster numbers. The latter task is denoted in figure 5.7 as an extra part of the process. The number of clusters k is the maximum number of generated clusters per clustering run, i.e., we generate a clustering model for each k from the range

specified by user. At the end of the clustering procedure, we have a set of clusters for each cluster number specified. This set is used by the user to identify a meaningful number of clusters k. The user analyzes results for each clustering run to find out which k provides the model with the best performance.

According to the concept of the unsupervised learning (clustering), we have to specify the model parameters on the training set. We test the performance of the model in the following way: we remove one instance from the training set and put it into the test set. After that, we apply the model to the test set while keeping the clustering parameters constant. This procedure is performed for each instance and the *mean error* r for each number k is computed. The mean error r implies the average number of incorrectly clustered instances. The minimum value of r refers to the model with the best performance.

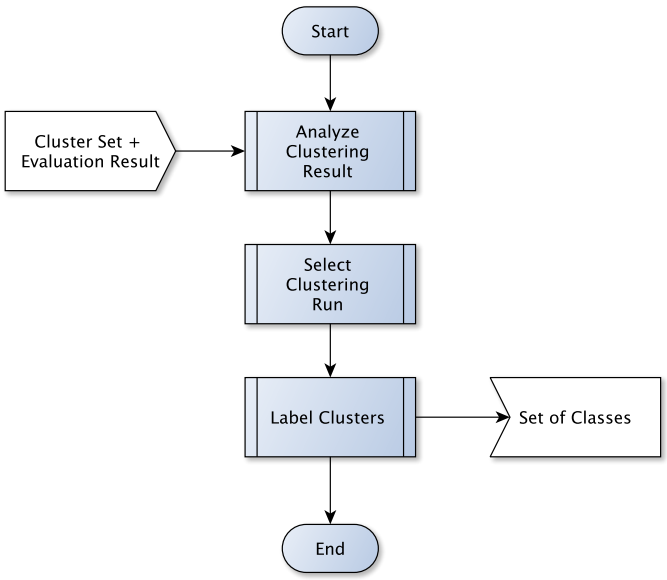

Figure 5.8: Overview of the classifier construction process: classification

The artifact of the described process is a set of clusters plus the performance value for each k. This artifact is used as input for the classification process illustrated in figure 5.8. Note that this process includes solely non-framework activities. The user decides which group of clusters corresponding to a cluster number should be used for prediction and performs classification by labeling clusters. Additionally, the user can characterize each cluster by a short description. The resulting set contains classes and classification model which is used for prediction in the next step.

5.2.3 Prediction

The last process deals with the prediction of project values by applying the classifier, which was constructed in the previous step, to the test set containing the ongoing project

(see figure 5.9). For this purpose, the system performs clustering of the training and test set, while keeping the parameters and the cluster number as defined by the classifier. This results in the assignment of the ongoing project to a class.

In order to predict project's behavior, we need identify the representative of the class first. To this end, we use the mean value of the cluster for each time frame. Finally, we estimates the future values of the test project by using the class representative.

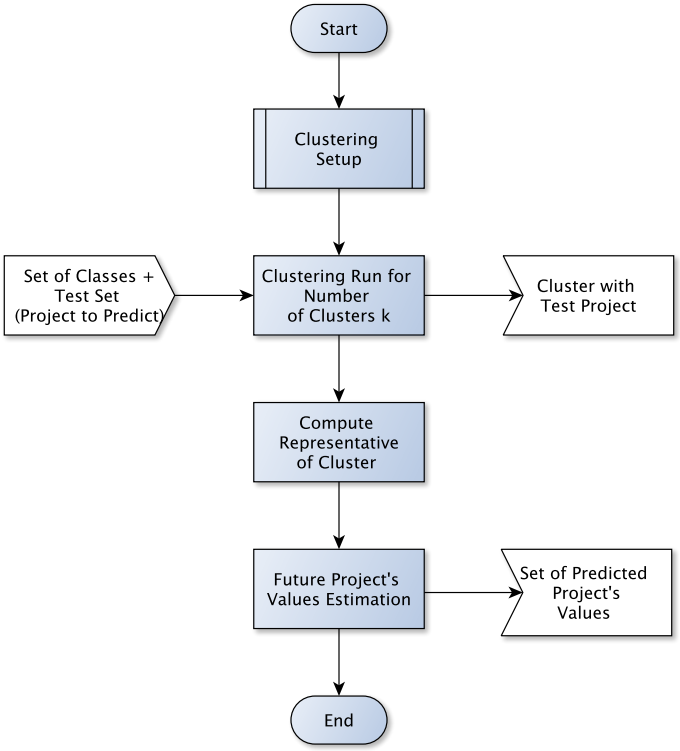

Figure 5.9: Overview of the classifier application: prediction

The estimated values are continuously compared to the actual values during project's progression. Thereby, the estimated values are replaced by the new actual values and the application of the classifier is triggered. Repetitive classification of the project instance might result to assigning the instance to a new class and rectification of estimating values is need.

5.3 GUI Prototyping

Following the objective of this thesis to evaluate our concept with industrial data, we need to design a software prototype, which, simultaneously, will be a proof of concept for the framework. In section 5.2, we modeled possible scenarios regarding functionality for the tool.

In this section, we create a *prototype* of the user interface. Prototyping is *"the method of brainstorming, designing, creating, testing, and communicating user interfaces"* [Sny03] and is widely used in software development for conducting usability studies before developing an actual software. We designed our prototype within a software tool supporting the paper prototyping technique. There exists a number of tools providing creation of screen mockups. A collection of tools for design paper prototypes, along with pros and cons, can be found in [Gui14]. We decided to use *Pensil*, an open-source GUI prototyping tool [Evo12].

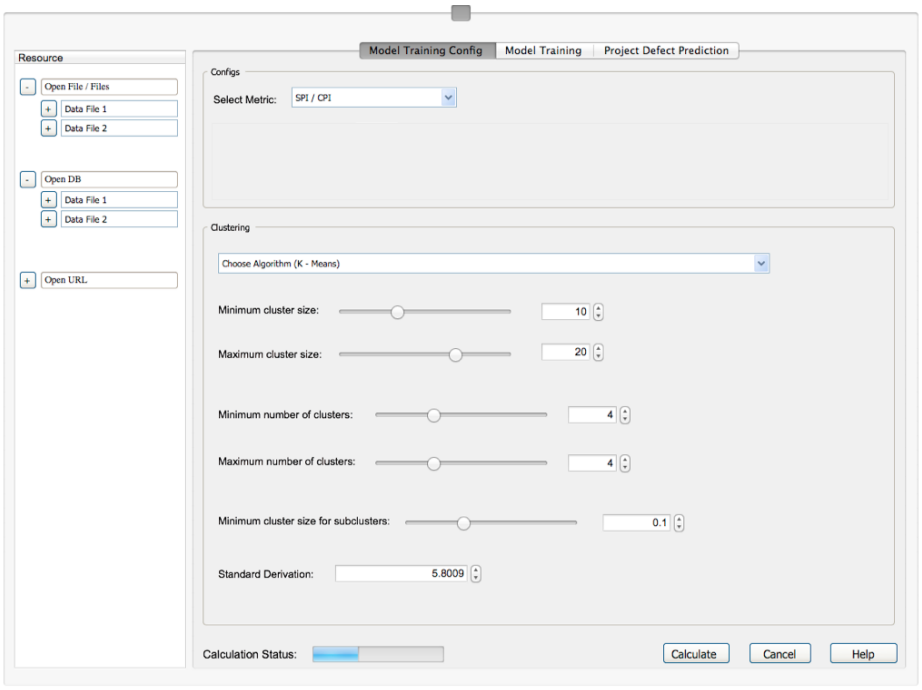

Figure 5.10: Screen mockup of data import and clustering setup page

The design of our prototype is based on the requirements defined in chapter 3. First, we need to build a web-based system supporting various import options (cf. requirement **FR1.4**). Figure 5.10 demonstrates the prototype of the import and clustering setup page. The current prototype offers three possibility to import the data into the framework: by opening a file, accessing a database, or using an URL.

According to the requirement **FR1.3** and considerations in section 5.2.2, the user has to configure several parameters before triggering the clustering procedure. The configuration consists of selecting the metric to be used for prediction. Further, we can set parameters to configure the clustering method: selecting the clustering algorithm, setting the cluster size (i.e., a maximum and a minimum number of elements in each cluster), and defining the range of cluster sizes (to performing a k-clustering for each number within this range; cf. section 5.2.2). Additionally, the user can see the calculation status, after they trigger the clustering process by pressing the button *"Calculate"*. This

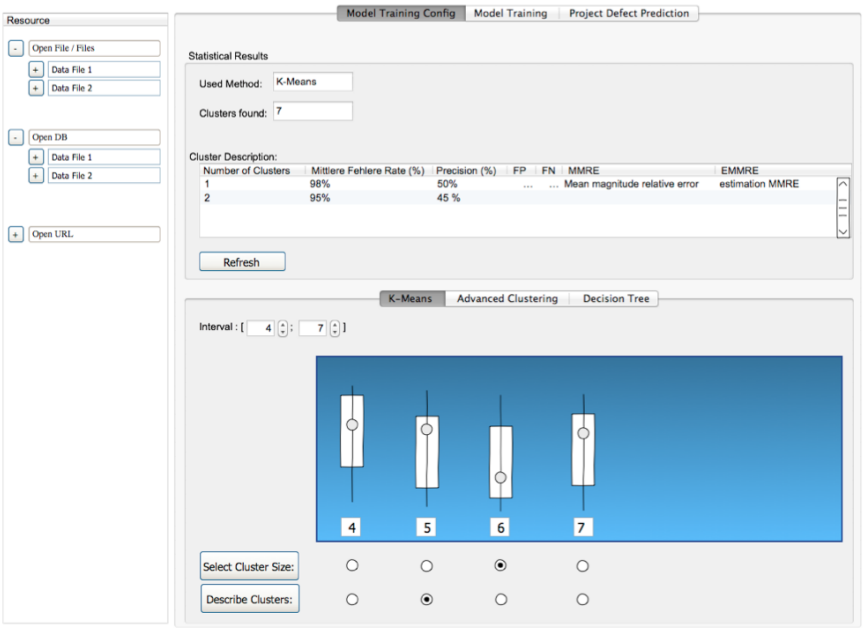

Figure 5.11: Screen mockup presenting clustering result page

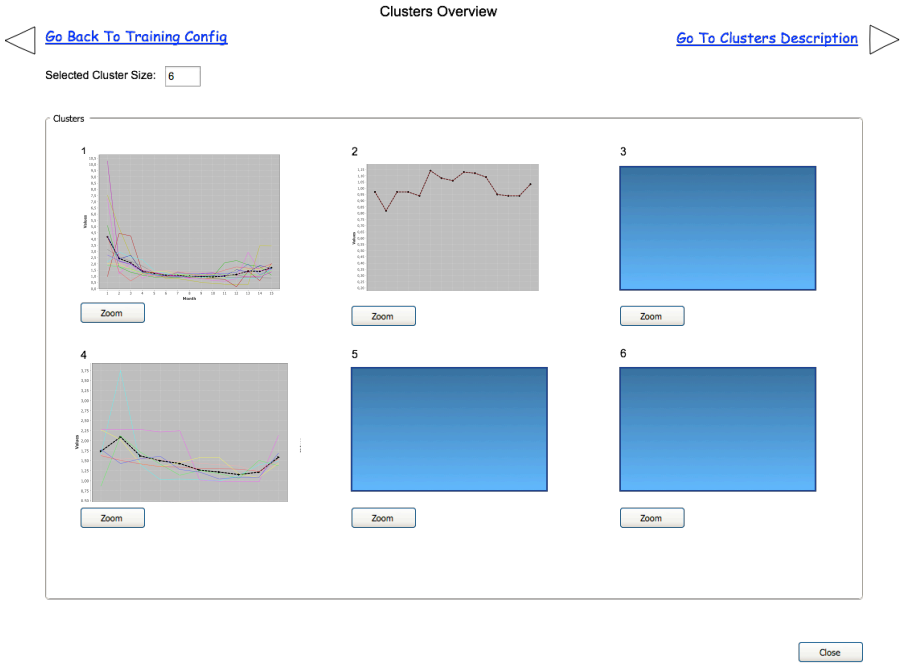

Figure 5.12: Screen mockup illustrating clusters overview page

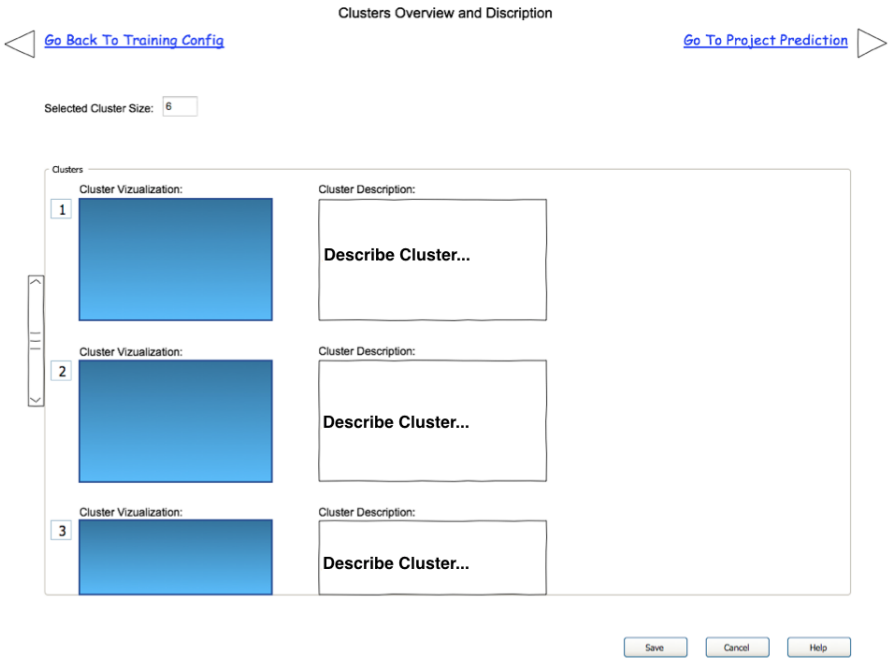

Figure 5.13: Screen mockup illustrating classification page

process can be aborted by pressing *"Cancel"*. A press on the button *"Help"* provides further information regarding the clustering parameters.

After the clustering procedure is initiated, the framework performs the clustering. When the generation of clusters is finished, the user is redirected to the next page displaying the calculation results depicted in figure 5.11. With respect to requirements FR160 and FR180, the statistical results of each clustering run are presented in tabular and graphical (box-and-whisker chart) forms. The box-and-whisker chart in figure 5.11 reflects mean errors of each k-clustering. In figure 5.12, the user has an overview of a selected cluster.

When the decision regarding meaningful clusters number is made, the user can start with the classification. To this end, the user can either click the link *"Go To Clusters Description"* at the top of the clusters overview page or activate the button *"Describe Clusters"* under the box-and-whisker chart. Both options lead the user to the *"Cluster Overview and Description"* page illustrated in figure 5.13). On this page, then, the user can label clusters with names as well as write short description for each cluster.

After the classification, the user can save the model with classes and trigger prediction using the link at the top of the *"Cluster Overview and Description"* page. In the prediction page shown in figure 5.14, the user is asked to open the ongoing project and specify the metric for the prediction. On the same page, the user can see prediction result, when calculation is completed. They can configure the visualization mode by picking a check-box on the screen. The user can choose either prediction line, prediction area,

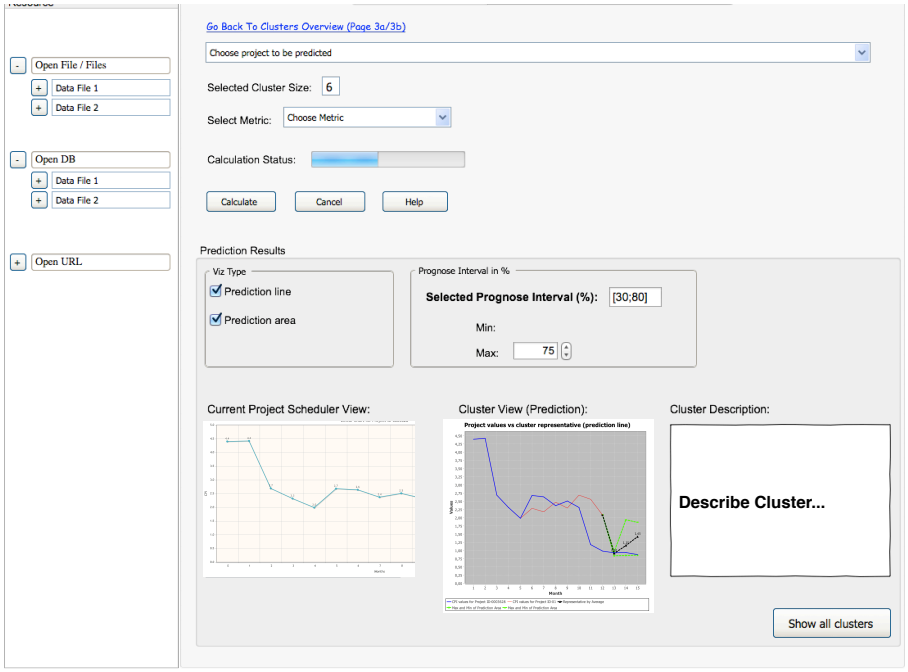

Figure 5.14: Screen mockup presenting prediction result

or both. The corresponding representation appears in the graph reflecting prediction result. Next to the graphs, representing estimated project's progress, the user can see the corresponding cluster and its description.

5.4 Domain Model

In this section, we explain the domain model of the PredictionTool. Figure 5.15 demonstrates the UML entity relationship diagram of the domain model. It includes all components needed to maintain the required functionality of the PredictionTool that is described this chapter. The domain model is logically separated into the Import component and the Clustering component.

The `Import` component contains all entities that are created during the import of data into the system: `Data Basis`, `Data Vector`, and `Data Value`. The data basis entity represents a general container of logical group of data, e.g., a set of projects comprising 'small', 'medium', and 'large' projects. Each data basis object possesses a unique name, a description, and a list of data vector objects (the projects). The data vector entity has two attributes: `EOM` and `MetricID`. The `EOM` attribute is used to reflect the unique name of each data vector according to which it can be found in the database, e.g., the project ID. The `MetricID` attribute stores the characteristic of the data vector according to which this vector should be assigned to what cluster in the

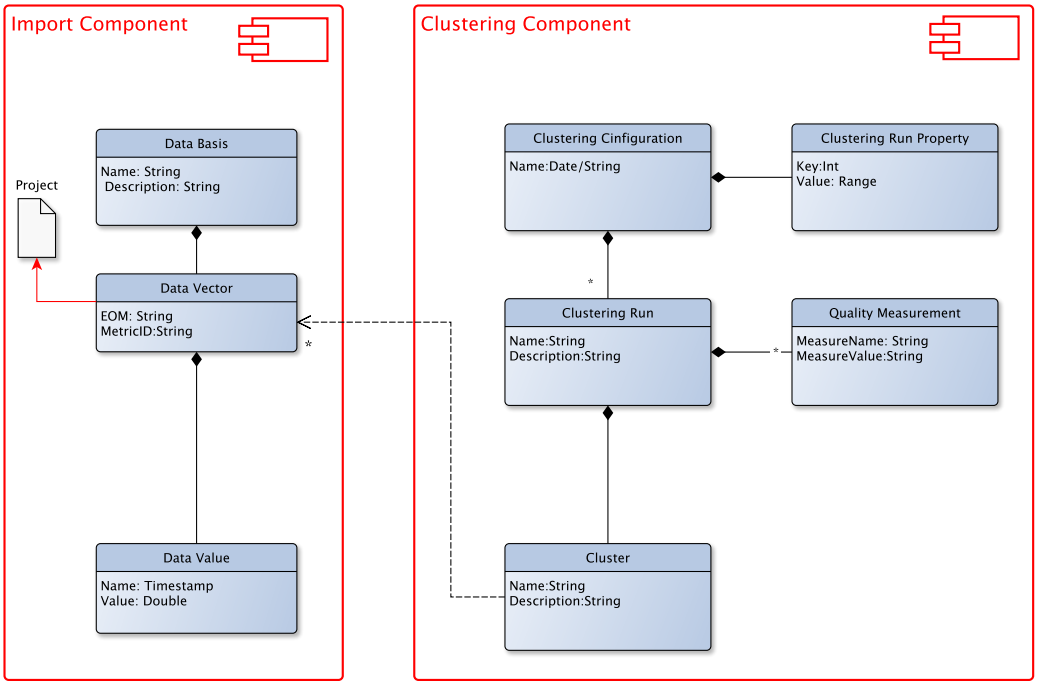

Figure 5.15: Architecture: the domain model

clustering process. Moreover, each data vector has a list of data values, which are the time series values measured for the certain metric (`MetricID`). Consequently, these data value objects must provide a timestamp with the corresponding value.

The `Clustering` component comprises of all entities that are created during configuration of the clusterer and the clustering process: `Cluster`, `Clustering Run`, `Clustering Configuration`, `Clustering Run Property`, and `Quality Measurement`. The clustering configuration entity has only the attribute `Name`, in order to store the date that implies when a particular configuration has been created. Additionally, clustering run configuration possesses a clustering run property and a list of clustering runs. The clustering run property entity encapsulates the information about the number of total clustering runs required, which is stored in the `Key` attribute, and the maximum clusters number for each ran, which is store in the `Value` attribute. The clustering run object is created when a clustering process is initiated for each maximum number of clusters. Clustering run is specified by a unique name and a description. In order to be able to store the performance of each clustering run according to a certain quality measurement, the quality measurement entity is used. It includes the attribute `MeasureName` to reflect the corresponding measurement method, and the attribute `MeasureValue` to store the evaluation result.

Note that each clustering run contains a set of clusters obtained after the clustering process. The cluster object can be identified according its unique name. Additionally,

the cluster entity has the description attribute needed for the classification. Each cluster refer to a set of assigned to it data vectors, which is denoted by the dashed arrow.

6 Realization

Contents

6.1 JavaEE Technologies . . . 53
6.1.1 Enterprise JavaBeans . . . 54
6.1.2 Java Server Faces . . . 54
6.2 Architecture . . . 55
6.2.1 Component Architecture . . . 55
6.2.2 Data Import Component . . . 57
6.2.3 Data Pre-Processing Component . . . 59
6.2.4 Clustering Component . . . 61
6.2.5 Classification and Prediction Components . . . 63
6.3 WEKA Java API . . . 64
6.3.1 The ARFF Format . . . 65
6.3.2 Accessing the Data . . . 66
6.3.3 Clustering with WEKA . . . 69
6.4 Visualization . . . 71
6.4.1 PrimeFaces Chart . . . 71
6.4.2 JFreeChart . . . 73
6.5 Evaluation of Clustering Performance . . . 73

In this chapter we describe the realization aspects of our approach. First, we describe the technologies that we integrated to implement the system in section 6.1. We list the main challenges of the component architecture and explain how we implemented them in our software prototype in section 6.2. Subsequently, we will go through each component and describe its main interfaces and hot-spots. After that we describe the implementation details of the integration of Weka Java API that we used as computational core for air approach.

Implementation of the visualization component as well as the tool applied are described in section 6.4. Finally, in section 6.5, we briefly explain the two-step procedure that we implemented to evaluate the clustering performance.

6.1 JavaEE Technologies

In this section, we give some background information for the techniques that we used to contract our software prototype. We start by describing the Enterprise JavaBeans (EJB) framework that we use for the back-end architecture in section 6.1.1. Then, in section 6.1.2, we explain Java Server Faces (JSF) that we use for the client component.

6.1.1 Enterprise JavaBeans

For our back-end architecture, we use the Enterprise JavaBeans (or shortly EJB) framework. In its specification, EJB is defined as *"an architecture for component-based transaction-oriented enterprise applications"* [SM09]. The EJB technology is widely applied in business software applications. The reason is that EJB offers many features for developing business systems: database accessing using different techniques (e.g., *Java Database Connectivity (JDBC) API*), integration with other applications through the Java EE Connector Architecture, etc.

Enterprise beans are server-side software components. There are two main server-side component types provided by EJB: *session* and *message-driven* beans. Session beans serve to implement application activities, e.g., adding elements, accessing a database, or calling other beans. Moreover, they act either as *stateless* or *stateful*. Stateless session beans are used to implement business logic that is performed by a single request without keeping the client-specific state among calls. Stateful session beans are used to model a process, which spans multiple user request. In contrast, message-driven beans are also intended to maintain actions, but can be called only implicitly by sending *messages*. For more information, see [RB10] and [SM09].

Starting with version 3.0, EJB uses *POJO* to avoid API coupling upon the classes. The acronym POJO stands for *Plain Old Java Object* and was defined by Fowler [Fow00]. POJOs refer to general Java objects that neither are inherited from any class nor implement any interface. However, to create enterprise beans from POJOs, we have to configure them within a deployment descriptor, e.g., XML files, or by annotation provided by EJB.

Enterprise beans are deployed within an *EJB Container*, which provides automatic generation, management, and destruction of EJBs, and thus, requires no intervention of the developer. Accessing the beans is further possible from the outside of the container, e.g., using SOAP or REST web services.

6.1.2 Java Server Faces

The front-end architecture of our DataMiningFramework is based on the Java Server Faces (or shortly JSF) technology. It is a server-side component framework for developing web applications written in Java. JSF offers various programming model and tag libraries for implementation of component tags that simplify creating and maintaining web applications with server-side user interfaces.

JSF technology provides separation of logic from presentation that allows autonomous development of each single part of the software application. The presentation layer is typically implemented in the form of XHTML-Pages. In order to represent logic layer, JSF uses *Managed Beans*, which are regular Java Bean classes (POJOs) that are registered by JSF using annotation *@ManagedBean*. Managed Beans represent a model for UI component and typically contain the getter and setter methods. Those can be accessed from the XHTML-Page.

A JSF application runs in a Java servlet container, where Java beans are automatically

instantiated as soon as they are needed by the application. The lifetime of the particular bean is also defined through JSF annotation. There exist four different lifetime scopes provided by JSF: The shortest is the request scope. Lifetime of such beans is equal to the lifetime of the HTTP-requests. After the HTTP-requests is complete, those beans are released. The view scope means that the beans with this scope live as long as the view to which they are bound. The third lifetime scope is the session scope. Here the bean is released first after an HTTP session is completed. The longest scope is the application scope. This means that there is only one bean for all users during performing the entire application.

More information about the JSF technology can be found in [Ora13].

6.2 Architecture

Design of the system's architecture is the next abstraction step in software development. Software architecture design is a sophisticated process and the resulting architecture should meet all technical and operational requirements. Bass et al. explains software architecture of a system as *"the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them"* [BCK03]. The main objective of the architecture is organizing components to provide specific functionality. Through the process of components organization, components are grouped according to their area of concern, within which each component is responsible for only one concern or functionality [Pat09].

In this section, we describe the architecture of our framework. We start with a coarse-grain component architecture and explain the main functionality of each component as well as the relationships among them. After that, we explain the architectural details of each component.

6.2.1 Component Architecture

In this section, we describe the system architecture that we derived from our requirements (cf. 3) and some general requirements for a software prototype (cf. [Pat09]). In brief, the software architecture is designed in compliance with the following three key principles:

Loose coupling between components. This can be achieved by separating the components w.r.t. their functionalities, which should overlap as little as possible. This implies, that the system supports extending and optimizing the functionality of each component without touching the source code of some other component. For instance, the loose coupling of back-end and front-end components enables replacement of the presentation layer technologies with little effort; e.g., we can replace JSF by Angular JS technology.

Abstract implementation of interfaces by providing, e.g., a *Facade*, on the top of each component. Additionally, abstraction of the common interfaces between

components can be achieved by defining interface types or abstract classes. This helps to achieve transparency of the system's architecture for an external viewer.

Established communication between components. This issue addresses a contract or an interface specification of components within the application. This imply that the relationship between components, modules, and functions as well as their usage must be explicit defined. A contract specifies all possible communication scenarios between components as well as the access of the internal functionality of each component. Further, it describes pre-conditions, post-conditions, side effects, etc. of those scenarios.

With respect to the principles described above and requirements **FR2.1** to **FR2.5**, we defined seven components in our coarse-grain architecture, which is depicted in figure 6.1. Each component provides a business interface, i.e., the access of each component is provided by the corresponding *facade*. This is realized by EJB's `FacadeLocal` interface, which is implemented by a concrete EJB `FacadeBean`.

On the top of the application, the API component `JSF API`, which is implemented using JSF, is responsible for the presentation function (UI) of the system. It includes mainly XHTML-pages and Managed Beans as described in section 6.1.2. Additionally, `JSF API` enables the import of data into the system. This task might be replaced in the future by a measurement system.

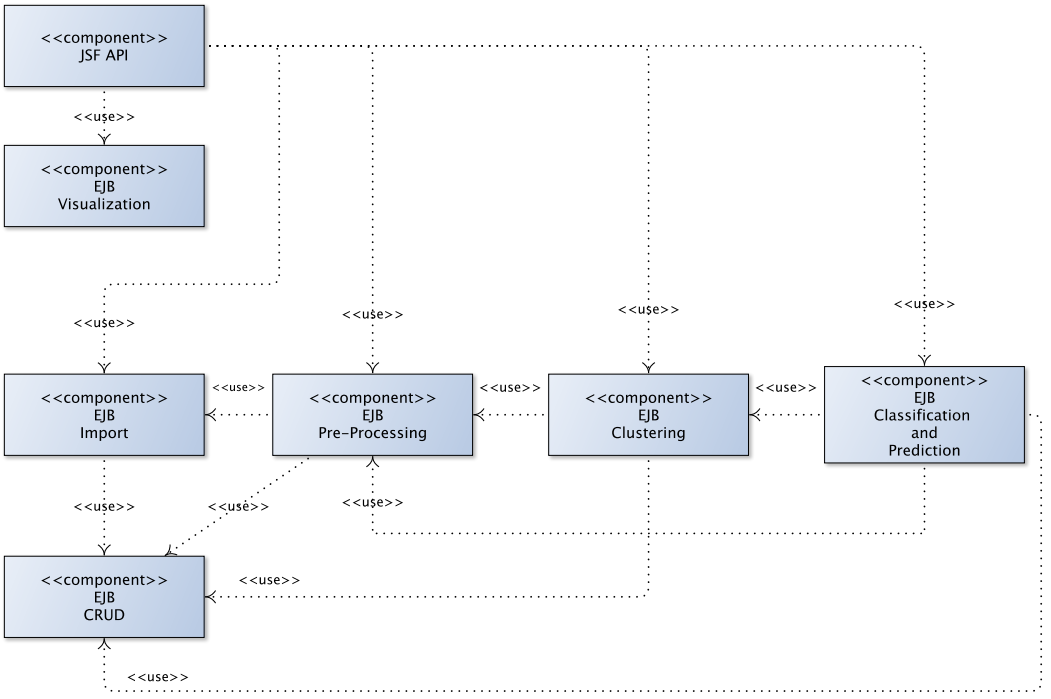

Figure 6.1: Architecture: components

Since JSF API encapsulates the GUI, it must communicate with other components in order to retrieve user's input and display the output of, e.g., the computation core of the system. JSF API invokes other components through the corresponding facade of each component and its implementation allows for the replacement of JSF by an other framework, e.g., Angular JS.

The Visualization component contains functionalities needed to create graphical elements, such as different charts to visualize the clustering and prediction output. In section 6.4, we give more information on implementation details and techniques used to create charts.

The four main steps of the knowledge discovering process are encapsulated in four components: Import, Pre-Processing, Clustering, and Classification and Prediction. Note that classification and prediction are combined into one single component, since in our approach, classification is a sub-function of prediction and is conducted by the user. The Classification and Prediction component uses the output of the Clustering component to conduct the classification and subsequently apply prediction.

Similar to the knowledge discovering process, the output of each of these components is the input for the next adjacent component. For instance, we use the pre-processing output as an input for starting the clustering process within the Clustering component. Interaction between components proceeds typically via facades to achieve flexible communication. Thus, if an alteration within a single component is required, the other components remain untouched. For the purpose to be able to customize the functionality of each component, the design of each component provides extension points (hot-spots) that are explained in sections 6.2.2, 6.2.3, and 6.2.4.

The last component, CRUD, represents the business layer of the application. The acronym CRUD states for **c**reate, **r**etrieve, **u**pdate, **d**elete functions of the back-end to handle application objects. Figure 6.2 illustrates the CRUD component, comprising the CRUD Facade, CRUD Controller, Domain Model and DAO classes. The facade CRUD Facade is the entry point for other components and forwards the control flow to the controller CRUD Controller, which validates the data and invokes the methods of the DAO classes. If an entity must be created, the controller uses Domain Model to create an entity via an entity factory. DAO is responsible for storing, updating, and deleting of the application using an Entity Manager.

Subsequently, we present the architecture of each component in more detail. Specifically, we observe the key components of the computation core, import, preprocessing, clustering, and classification and prediction.

6.2.2 Data Import Component

Figure 6.3 illustrates the Import component of the DataMiningFramework composing the DataImportFacadeLocal interface and the implementing DataImportFacadeBean class. DataImportFacadeBean delegates method calls to DataImportController which provides a particular implementation of the

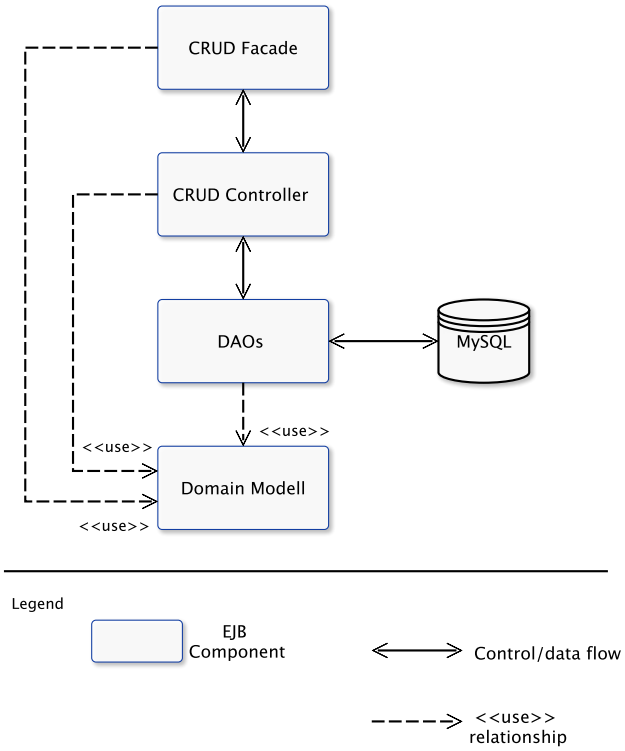

Figure 6.2: Architecture: overview of CRUD component

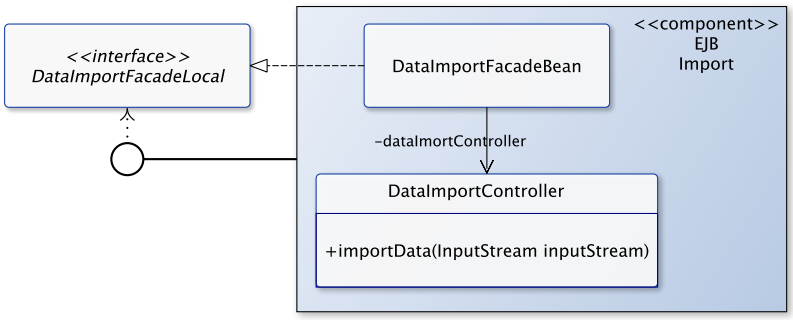

Figure 6.3: Architecture: Data import component

method `importData` that is used to read the data from an input stream (delegated from `JSF API`) and transform it into Java Objects.

Concerning the requirements **FR1.4.1** and **FR1.4.2**, the DataMiningFramework must offer the import of data from an Excel file as well as the data import from an SQL database. Currently the DataMiningFramework supports only Excel format, but as shown in figure 6.4, the architectural design of the system anticipates future

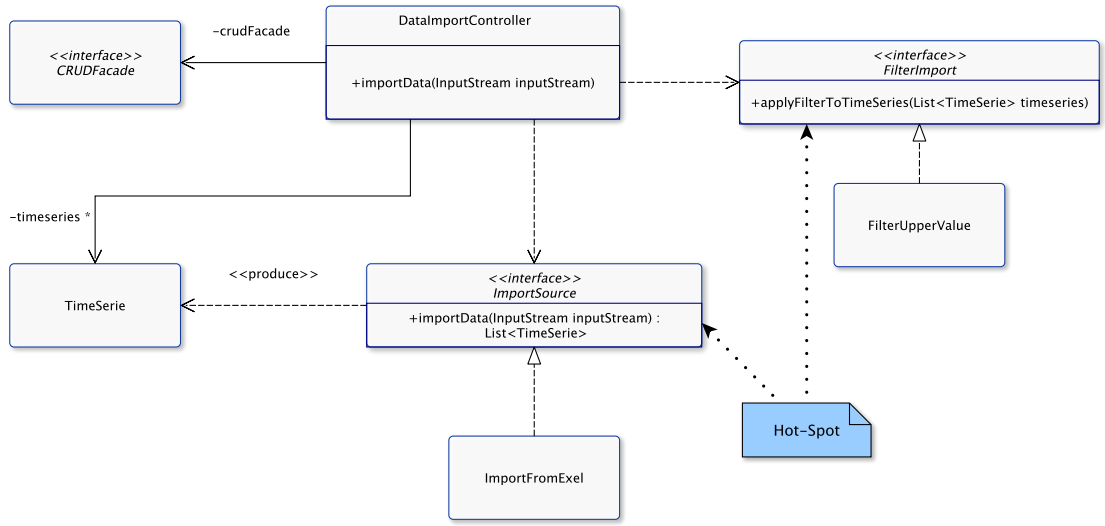

Figure 6.4: Architecture: Data import component with classes

implementation of additional import possibilities via an extension point (hot-spot). This extension point is represented by the interface of the type `ImportSource` comprising the hook method `importData`.

First, the imported data is transformed into Java objects representing time series in order to filter the imported data. Alike to the import format issue, the architecture of the system provides the `FilterImport` interface; each filter class implementing the `FilterImport` interface realizes a particular implementation of the filter function. For instance, the class diagram in figure 6.4 shows a filter `FilterUpperValue` that discards data with values exceeding a predefined threshold.

After the import and filter activities are finished, the time series are transformed to `DataVector` entities and are persisted in the database via the CRUD component.

6.2.3 Data Pre-Processing Component

As shown in figure 6.1, the next step after the import is data preparation in the `Pre-Processing` component. Here, the data is cleaned from outliers and data noise, and normalized in order to take a form suitable for the clustering process. As stated before, all method calls are passing through the facade `PreProcessingFacadeLocal`, which is implemented by `PreProcessingFacadeBean`, and are then delegated to `PreProcessingController` that manages the pre-processing tasks.

Figure 6.6 demonstrates the architecture of the `PreProcessing` component. Within this *prove of concept*, we assumed that the pre-processing task can be triggered by the user as well. The control flow, initiated in the `JSF API` component, is forwarded through `PreProcessingFacadeBean` to `PreProcessingControllerBean`. Then, the controller invokes the `PreprocessingInitialization` class that initializes

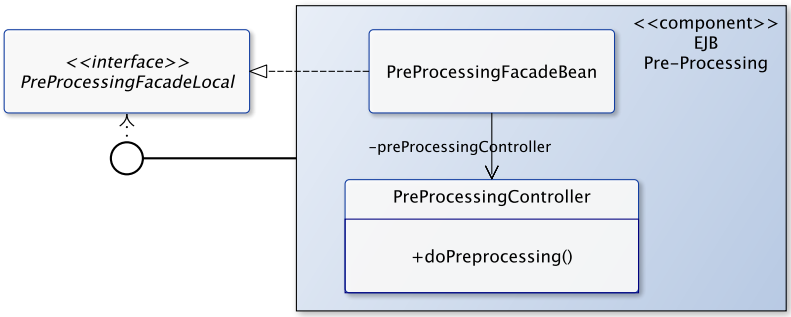

Figure 6.5: Architecture: Pre-processing component

pre-processing steps and allows to combine all available pre-processing steps in custom ordering. `PreProcessingController` processes those steps in the order defined by a list.

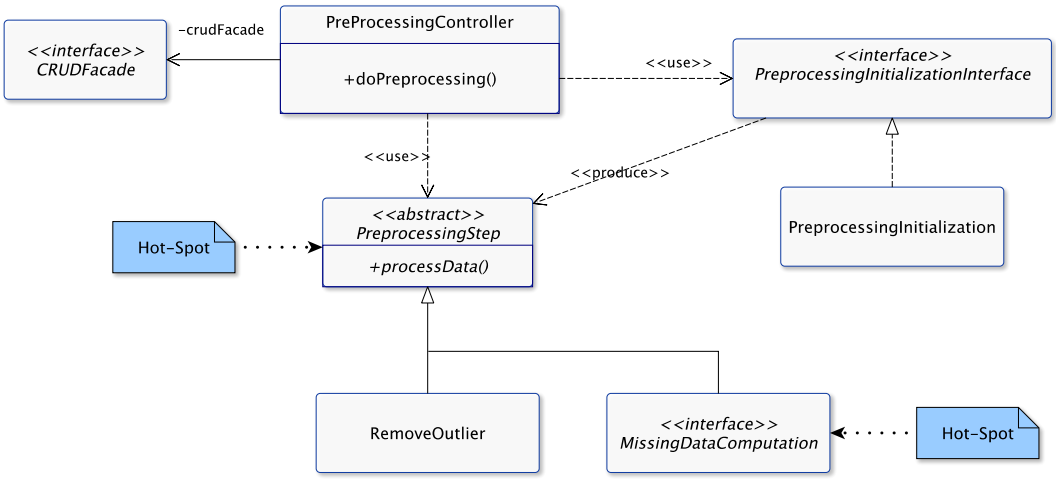

Figure 6.6: Architecture: Pre-processing component with classes

To allow alteration of pre-processing strategy, we created two additionally hot-spots in this component: a hot-spot for the regression method that is applied to compute missing values (see figure 6.7) and a hot-spot to allow re-creation and exchange of pre-processing steps. The former hot-spot is defined by the interface `RegressionMethod`, which is currently implemented by three different regression approaches. The latter hot-spot is defined by the hook method `processData` in the abstract class `PreprocessingStep`.

When `PreProcessingController` finishes the pre-processing, i.e., all steps in the pre-processing list were executed, the processed data is, then, persisted using the `CRUD` component.

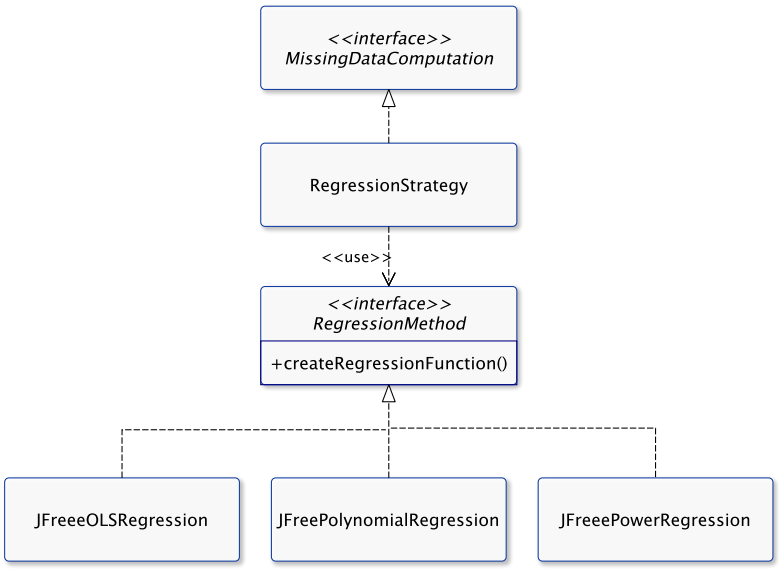

Figure 6.7: Architecture: Missing data computation hot-spot

6.2.4 Clustering Component

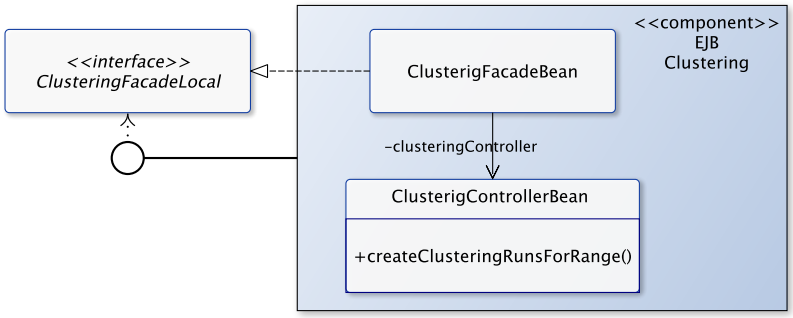

Figure 6.8: Architecture: Clustering component

Next, after the data is pre-processed, the data is ready for the clustering process. Figures 6.8 and 6.9 illustrate the structure of the `Clustering` component. After the method call is forwarded through `ClusteringFacade` to `ClusteringControllerBean`, the class invokes `SetupCluster` to trigger initial activities needed to perform clustering. This activities include invocation of additional pre-processing methods, if required, transformation of the data into the appropriate form supported by the clustering tool, as well as specifying of the clustering algorithm that will be applied.

When clustering is finished, the `SetupClustering` class performs interpretation of the clustering result, which is transformed it into a Java object of

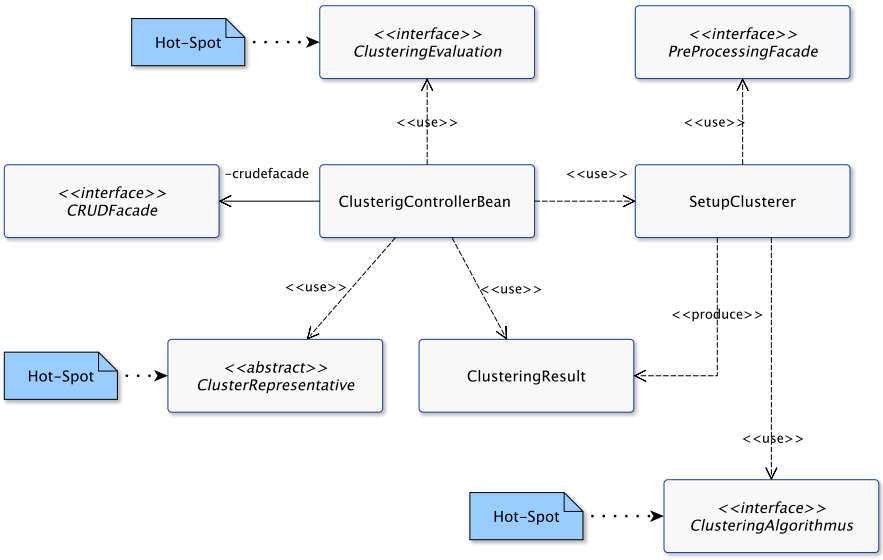

Figure 6.9: Architecture: Clustering component with classes

type `ClusteringResult`. `ClusteringControllerBean` uses the information encapsulated by the `ClusteringResult` object to create corresponding domain model objects with the obtained information. Moreover, the controller ensures that the `ClusteringRun` object and the list of corresponding cluster objects are created after each clustering process. Since the controller receives the list of maximum clusters numbers for each iteration, it initiates creation of clustering runs by calling the methods of `CRUDFacade`.

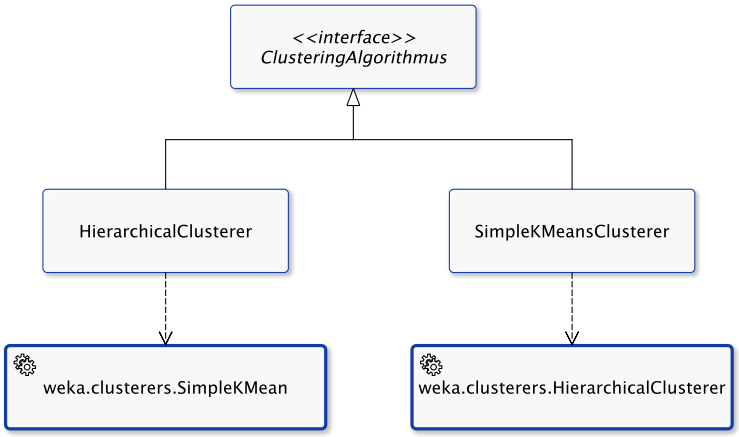

Figure 6.10: Architecture: Clusterer algorithm hot-spot

The `Clustering` component specifies two hot-spots. First, a hot-spot aiming

for the interchangeability of the clustering algorithm and is realized by the interface `ClusteringAlgorithmus` (cf. figure 6.10). The second hot-spot is the abstract class `ClusterRepresentative` that provides different methods to identify cluster representatives (see figure 6.11); e.g., a cluster representative that is computed using the average value of all vector values of the cluster.

A cluster representative is, then, used to build a prediction line or a prediction area for a certain project. This task is implemented by the `Classification and Prediction` component, which we explaine in the next section.

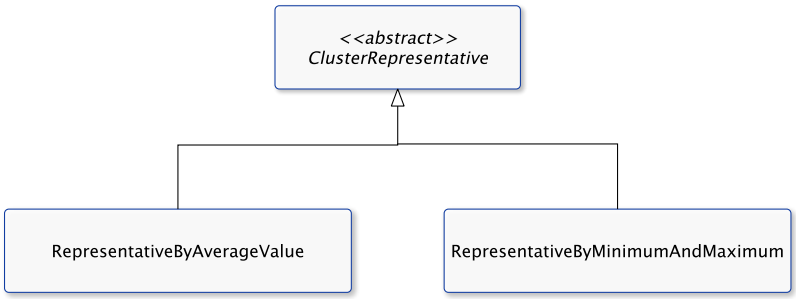

Figure 6.11: Architecture: Cluster representative hot-spot

6.2.5 Classification and Prediction Components

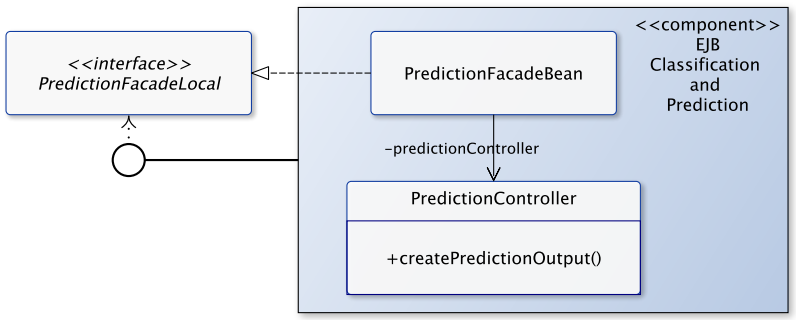

Figure 6.12: Architecture: Classification and Prediction component

When the clustering is accomplished, the user can initiate the classification and prediction process. To this end, the user is presented with the evaluation results for each clustering iteration (number of clusters) and can start the process by selecting the clustering with the minimal error for incorrectly clustered instances. First, the user is asked to label each cluster of the selected clusters set. To this end, the DataMiningFramework provides the possibility to characterize each identified cluster in form of textual description. The classification result, i.e., clusters description, is stored

in the database. Then, the prediction process is started by choosing the project that wants to be predicted.

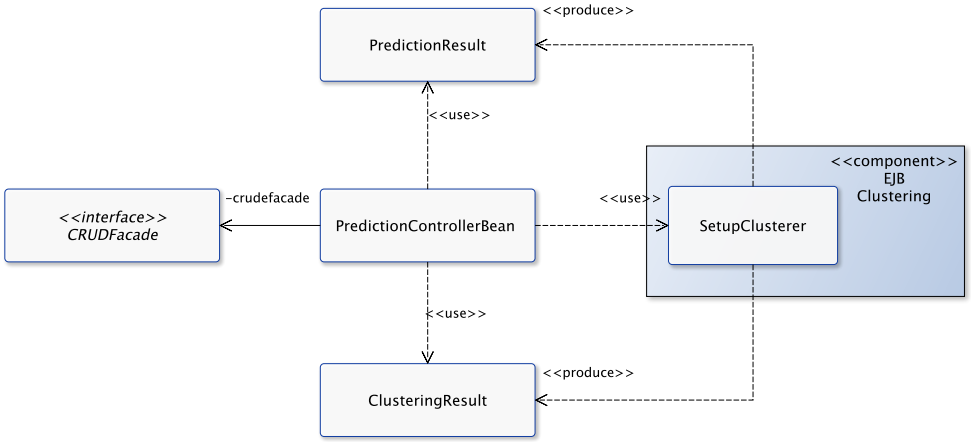

Figure 6.13: Architecture: Prediction component with classes

Similar to the other components, the access of the `Classification and Prediction` component is realized through the `PredictionFacadeLocal` interface which delegates all method calls to the `PredictionControllerBean` class. `PredictionControllerBean` initiates clustering of the *train-set*, i.e., all project used for the previous clustering (model training), for a selected number of clusters by calling the corresponding method of the `SetupClusterer` class. Then, `SetupClusterer` generates the `Clusterer` and the `ClusteringResult` object. Each `ClusteringResult` includes the cluster index as well as the list of the vectors that are assigned to that cluster. `ClusteringResult` together with the `Clusterer` and the *test-set* (project to be predicted) is used in the prediction procedure. This procedure is conducted by calling the `predictionOutput` method implemented in the `SetupClusterer` class. The method returns the `PredictionResult` object, that provides the index of the cluster to which the project to be predicted is assigned. According to this index, the underline cluster can be found in the database.

Once the cluster for the current project is identified, the DataMiningFramework computes the cluster representative that is used by the `Vizualization` component to create the prediction line for the project. We describe the visualization methods used and the obtained results in section 6.4.

6.3 WEKA Java API

In this work, we use the WEKA Java API Version 3.6.6. The integration of the library into the project is performed via a Maven dependency added to the pom.xml file as shown in 6.1.

```
1         <dependency>
2             <groupId>nz.ac.waikato.cms.weka</groupId>
3             <artifactId>weka-stable</artifactId>
4             <version>3.6.6</version>
5         </dependency>
```

Source Code 6.1: WEKA Maven dependency

Subsequently, in section 6.3.1, we describe the data format that is required by the WEKA library. After that, in section 6.3.2 we explain how the data is loaded into the system in order to be used by the WEKA API. Last, we address clustering in WEKA and explain the main challenges of using the WEKA API in section 6.3.3.

6.3.1 The ARFF Format

A set of data objects and their attributes is encapsulated in the data set in data mining. The data set in WEKA is represented by a collection of instances that are implemented by the class `weka.core.Instances`. The representation of this class is stored as an ARFF (Attribute-Relation File Format) file. An example of an ARFF file content is given in 6.2. It consists of two sections: the header and the data section.

```
1  @attribute Eom {ID-0005082,ID-0003162,ID-0003665,ID-0003533,ID-0005565}
2  @attribute Month_1 numeric
3  @attribute Month_2 numeric
4  ...
5  @attribute Month_10 numeric
6
7  @data
8  ID-0005082,2.24,2.11,1.95,1.78,1.64,0.97,0.9886,0.0823,0.762,0.53
9  ID-0003162,11.4,4.41,1.84,1.68,1.19,1.32,1.31,1.05,1.14,5.442556
10 ID-0003665,1.35,1.24,1.2,0.9,0.87,0.83,0.92,0.87,0.782857,0.782857
11 ...
```

Source Code 6.2: The WEKA Attribute-Relation File Format (ARFF)

In the header of the ARFF file, the attributes and their types are defined (cf. lines 1-11 in listing 6.2). Each attribute declaration consist of the `@attribute` keyword, followed by a unique attribute name and its type; i.e, each header line is of the form `@attribute <attribute-name> <datatype>`. The order of each attribute indicates position of the corresponding column in the database. The types of the attributes are either numeric, nominal, string, date, or relational and are declared as follows:

Numeric attribute is used to declare real and integer numbers and are of following format: `@attribute <attribute-name> numeric`

Nominal attributes are predefined values of the form `@attribute <attribute-name> {<nominal-name1>, <nominal-name2>, ... }`

String attributes enable to define attributes with arbitrary textual values and are declared in the the form: `@attribute <attribute-name> string`

Date attributes are used to specify a date and its format: `@attribute <attribute-name> date [<date-format>]`. Here, `<date-format>` is optional with a default value `<yyyy-MM-dd HH:mm:ss>`

Relational attributes are declared in the form:

```
@attribute <name> relational
<further attribute definitions>
@end <name>
```

In the example 6.2, `Eom` is a nominal attribute and `Month_n` are numeric attributes.

The ARFF data section starts with `@data` statement (cf. line 13 in listing 6.2) and is the declaration of instances (cf. lines 14-19 in listing 6.2), where each instance is represented by a single line. The attribute values of an instance are separated through commas and must appear in the same order as specified in the header section.

6.3.2 Accessing the Data

In order to start processing some data with WEKA, we need to transform it into the required format for WEKA. Typically, WEAK uses the ARFF format (cf. section 6.3.1, however, other formats, e.g., XRFF or CSV, are supported by WEKA. The WEKA API offers two possibilities to obtain data in the required format: load the data from source (specifically, from a file or a database) or create the data objects on-the-fly in memory.

Load from File

In case of loading data from a file, WEKA offers two types of loading: The first type of loader is defined by the file's extensions .arff, .csv, and .xrff. In all other cases, the user must use a loader for the corresponding file extension, e.g., `JSONLoader`, `C45Loader`, `SVMLightLoader`, etc. All available loaders are comprised in the package `weka.core.converters`.

Source code 6.3 demonstrates the reading of data from the default file types (ARFF, CSV, XRFF), using the `DataSource` class.

```
1 import weka.core.converters.ConverterUtils.DataSource;
2 import weka.core.Instances;
3 ...
4 Instances data1 = DataSource.read("/some/where/dataset.arff");
5 Instances data2 = DataSource.read("/some/where/dataset.csv");
6 Instances data3 = DataSource.read("/some/where/dataset.xrff");
7
8 Instances data1 = source.getDataSet();
```

Source Code 6.3: WEKA Data Loader: Loading with the DataSource class

Source code 6.4 shows how data in JSON-format is loaded by using `JSONLoader`.

```
1 import weka.core.converters.JSONLoader;
2 import weka.core.Instances;
```

```
3 import java.io.File;
4 ...
5 JSONLoader loader = new JSONLoader();
6 loader.setSource(new File("/some/where/some.data"));
7 Instances data = loader.getDataSet();
```

Source Code 6.4: WEKA Data Loader: Loading with JSONLoader

Load from Database

Using the class `weka.core.converters.DatabaseLoader`, we can loading data from a databases. A `DatabaseLoader` object queries the instances within a database either in batch or incremental mode. In the former mode the whole dataset is loaded from the database at once. The latter loading mode queries only a single row (instance) per query.

```
1 import weka.core.Instances;
2 import weka.core.converters.DatabaseLoader;
3 ...
4 DatabaseLoader loader = new DatabaseLoader();
5 loader.setSource("jdbc_url", "the_user", "the_password");
6 loader.setQuery("select * from whatsoever");
7 Instances data = loader.getDataSet();
```

Source Code 6.5: WEKA Data Loader: DatabaseLoader used in batch mode

Source code 6.5 and source code 6.6 illustrate the difference between loading in batch and incremental mode, respectively. Since not all databases support an incremental querying, WEKA additionally provides a "pseudo-incremental" mode, where the data fully loaded into memory first and after that is provided row-by-row.

```
1 import weka.core.Instance;
2 import weka.core.Instances;
3 import weka.core.converters.DatabaseLoader;
4 ...
5 DatabaseLoader loader = new DatabaseLoader();
6 loader.setSource("jdbc_url", "the_user", "the_password");
7 loader.setQuery("select * from whatsoever");
8 Instances structure = loader.getStructure();
9 Instances data = new Instances(structure);
10 Instance inst;
11 while ((inst = loader.getNextInstance(structure)) != null)
12   data.add(inst);
```

Source Code 6.6: WEKA Data Loader: DatabaseLoader used in incremental mode

Create the Data in Memory

In contrast to loading the data from a source, we can create the data in-memory on-the-fly using the class `weka.core.Instances`. In this way, the WEKA ARFF

data can be generated directly with the Java objects and a re-import of the data from a file or a database is not necessary.

Source code 6.7 shows how a list of the `DataVector` objects is converted into ARFF format that is needed for further clustering process in WEKA. You can see an example for the resulting ARFF format in example 6.2. The process of converting the data is conducted in two steps: first, the attributes must be defined, and second, the data is added row by row.

```
1  import weka.core.Attribute;
2  import weka.core.FastVector;
3  import weka.core.Instance;
4  import weka.core.Instances;
5  ...
6  List<DataVector>  vectors_list = new ArrayList<DataVector>();
7  vectors_list.addAll(vectors);
8  FastVector nominal_attrs = new FastVector();
9  for (DataVector vector : vectors_list){
10     nominal_attrs.addElement(vector.getEom());
11 }
12 attrs.addElement(new Attribute("Eom", nominal_attrs));
13
14 List<DataValue> values_list = new ArrayList<DataValue>();
15 DataVector first_element = (vectors_list.get(0));
16 values_list.addAll(first_element.getDataValues());
17 int vector_size = values_list.size();
18 for(int i=1; i<vector_size + 1; i++ ){
19     Attribute a = new Attribute("Month_" + i);
20     attrs.addElement(a);
21 }
22 Instances dataset = new Instances("Clustering of CPIs ",
23                 attrs, 500);
24 for(DataVector vector : vectors_list){
25     Instance instance = new Instance(cluster_size + 1);
26     String eom = vector.getEom();
27     instance.setValue((Attribute)attrs.elementAt(0), eom);
28     values_list = new ArrayList<DataValue>();
29     values_list.addAll(vector.getDataValues());
30     int i = 1;
31     for(DataValue dataValue : values_list){
32         instance.setValue((Attribute)attrs.elementAt(i), dataValue.getValue());
33         i++;
34     }
35     dataset.add(instance);
36 }
37 dataset.setClassIndex(dataset.numAttributes() - 1);
```

Source Code 6.7: Converting Java objects into ARFF format

For creation of attributes we use the `weka.core.Attribute` class. In order to cluster the `DataVector` objects, we use numeric attributes for double values and a

nominal attribute for all Eoms to enable identification for each `DataVector` after clustering. Since the nominal attribute is in this case not just a value but an array of strings, we use the `weka.core.FastVector` class.

After defining the attributes, for each entry in the `DataVector` list we create an `Instance` object with corresponding nominal attribute (Eom) and a set of numeric (double) values. Then, this instance is added to a dataset. which is a set of instances. The obtained dataset is used as the input for clustering in WEKA.

Discussion

In this work, we generate the data *in-memory* by converting the imported data into the ARFF format as described in the previous sub-section. We want to briefly discuss the advantages of the *in-memory* approach over the data loading from a databases or a file. Here, the main argument is that the transforming of the data in-memory decouples the domain-specific import and data objects from the used technology (here, WEKA). Thus, providing modularity of the system and independence of the tools used to perform, e.g., clustering operations.

For instance, assume that we decide to generate the data for clustering with WEKA by loading from a database. In other words, the WEKA loader imports the data and transforms it into WEKA instances. Hence, all DataMiningFramework procedures must be designed to process data in this WEKA-specific format, or the data must be transformed back and forth within the DataMiningFramework. The first option restricts the DataMiningFramework by using only the WEKA tool and the second issue implies computational overhead by performing multiple transformations. Hence, if we decide to use another technology for the clustering and prediction, we effort for the substitution is major. In contrast, creating the ARFF data in-memory allows generating the WEKA-specific data from the domain objects, and thus, decoupling the data import from the clustering tool.

6.3.3 Clustering with WEKA

The main reason we use WEKA is that the WEKA Java API offers various clustering algorithms. Everything needed to build and evaluate a clusterer is included in the `weka.clusterers` package. Clustering with WEKA is performed either in batch-manner or incremental. The former allows clustering of the whole dataset at once. In constrast, the incremental clustering updates its clustering model step by step as data is read. However, only a small number of clustering algorithms in WEKA supports incremental clustering, e.g., the Cobweb algorithms. Thus, in this work, we use only batch clustering. It is conducted in two steps: first, set the options, and second, build the model with the training data.

To evaluate the clustering result, WEKA offers the interface `weka.clusterers.ClusterEvaluation`, which is implemented by various cluster evaluation methods specific to the used clustering method. Here, a clustering

evaluation comprises a classification of cluster instances w.r.t. a cluster representative and a report of how many elements (absolutely and relatively) are in each cluster.

Additionally, WEKA can perform classes-to-clusters evaluation for probabilistic clustering methods (e.g., EM clustering). In general, such evaluation of clusterer is performed in three steps: First, we have to copy the training set of data containing the class attribute and remove the class attributes by applying the Remove filter of the weka.filters.unsupervised.attribute.Remove class. In our example, we have to apply the Remove filter to the Eom-attribute (see step 1 in source code 6.8). Once the filter is applied to the training set, we create the clusterer based on the filtered data in the second step (see step 2 in source code 6.8). In the third step, we create the ClusterEvaluation and apply it to the *original* training set in order to evaluate the clusterer (see step 3 in source code 6.8).

```
1  // 1. Create copy of train set without eom-attribute => dataClusterer
2  Instances train_set = ... // from somewhere
3  train_set.setClassIndex(0);
4  Remove filter = new Remove();
5  filter.setAttributeIndices("" + (train_set.classIndex() + 1));
6  filter.setInputFormat(train_set);
7  Instances dataClusterer = Filter.useFilter(train_set, filter);
8
9  // 2. Train clusterer with dataClusterer
10 EM clusterer = new EM();
11 // Set options for clusterer if necessary
12 ...
13 clusterer.buildClusterer(dataClusterer);
14
15 // 3. Create ClusterEvaluation and perform evaluation of clusterer
16 ClusterEvaluation eval = new ClusterEvaluation();
17 eval.setClusterer(clusterer);
18 eval.evaluateClusterer(train_set);
19 // Return evaluation result as string
20 result = eval.clusterResultsToString();
```

Source Code 6.8: Classes-to-clusters evaluation of EM clusterer

The last statement in source code 6.8 serves to return the evaluation result as string. An example of an evaluation result for an an EM clustering is shown in listing 6.9.

```
1  Clustered Instances
2  0       4 ( 29 %)
3  1      10 ( 71 %)
4
5  Log likelihood: -8.36599
6
7  Class attribute: play
8  Classes to Clusters:
9
10  0 1  <-- assigned to cluster
```

```
11   2 7 | yes
12   2 3 | no
13
14 Cluster 0 <-- no
15 Cluster 1 <-- yes
16 Incorrectly clustered instances : 5.0  35.7143  %
```

Source Code 6.9: Evaluation Result

6.4 Visualization

In this section, we describe the data visualization for the DataMiningFramework and the used library. According to requirement **FR1.8**, besides the statistical representation, our prototype must provide the graphical representation of clustering and predication results. Thus, we have to decide what visualization library to use in order to render the diagrams.

The visualization library must provide several features that satisfy objectives and requirements of the DataMiningFramework. First, it must be compatible and easily integrable with the client layer of the DataMiningFramework, namely, with the JSF component. This implies that the library must be written in Java. Second, it must provide rendering of all diagrams required by the DataMiningFramework. Hence, the selected library must support various types of charts, including combined charts as well as the customization of their labels. Moreover, the library must be able to operate with time series for diagram and it must be able to the render diagrams independently from the internet browser.

According to the requirements for a visualization library listed above, we chose two: *PrimeFaces Chart* and *JFreeChart*. PrimeFaces is an open-source JSF component suite, and thus, is obviously compatible with JSF applications. The Chart component of PrimeFaces allows to create various types of chart and dynamically render them in an JSF page. However, it does not provides combinations of charts and restricts alteration of the diagrams for the developer.

JFreeChart is an open-source library for the Java written applications and provides a various number of diagrams and their combinations, annotations, export to PNG, JPEG, PDF, SVG, etc. Furthermore, the JFreeChart library provides implementation of several regression methods that can be applied in-memory without a visualization. The essential drawback of JFreeChart for our prototype is that the diagrams created by JFreeChart do not provide interactivity, since they are rendered into the JSF page as a simple buffer image.

6.4.1 PrimeFaces Chart

A PrimeFaces chart can be implemented directly in the corresponding `@ManagedBean` of the `JSF API` component. Each PrimeFaces chart is an object of the type `LineChartModel` that is called from an XHTML-page (cf. source code 6.10). Source

code 6.11 illustrates an example where the PrimeFaces Chart library is used to visualize a `DataVector` object, creating a `LineChartModel`. The resulting chart is depicted in figure 6.14.

```
<p:chart type="line" model="#{projectsOverviewBean.lineChartModel}"
            style="height:700px;" />
```

Source Code 6.10: A line chart using PrimeFaces, XHTML

```
private String createLineModels(DataVector vector) {
        this.lineChartModel = initLinearModel(vector);
        this.lineChartModel.setTitle("Linear Chart for Project "
            + vector.getEom());
        this.lineChartModel.setLegendPosition("e");
        this.lineChartModel.setShowPointLabels(true);
        this.lineChartModel.getAxes().put(AxisType.X,
            new CategoryAxis("Months"));
        Axis yAxis = lineChartModel.getAxis(AxisType.Y);
        yAxis.setLabel("CPI");
        yAxis.setMin(0);
        yAxis.setMax(5);
    }
    return "showVectorChart.xhtml";
}
```

Source Code 6.11: A line chart using PrimeFaces, @ManagedBean

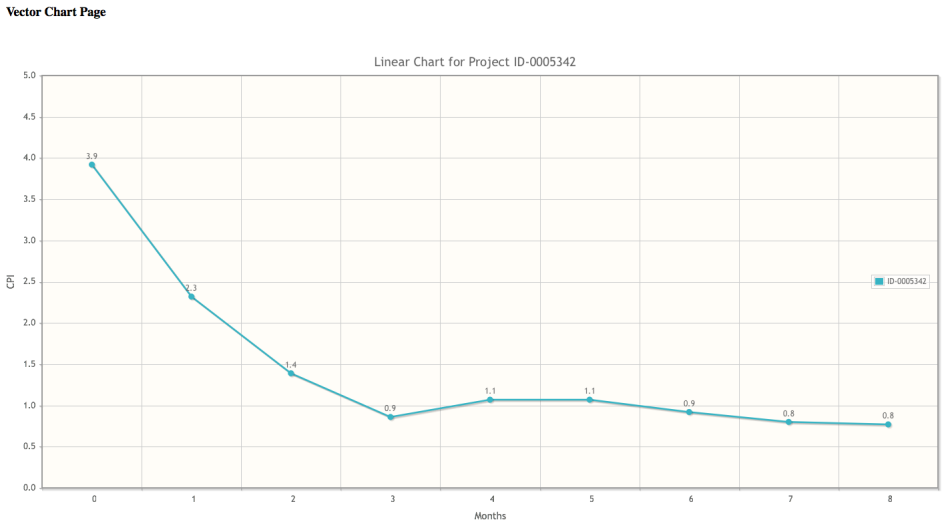

Figure 6.14: Resulting PrimeFaces line chart

6.4.2 JFreeChart

Source code 6.12 shows how to create a chart for a cluster using JFreeChart. In order to create a JFreeChart diagram, the given time series must be transformed into a `XYDataset` first (lines 3-4). The example creates a PNG file that illustrates all projects assigned to the cluster and the cluster representative as `XYLineChart`. Figure 6.15 illustrates the resulting diagram.

```
1  public File getClusterAsChart(Cluster cluster)
2              throws IOException, NotFoundException{
3  XYDataset lines =  ConvertDataIntoXYDataset
4      .convertClusteredProjectsIntoXYDataset(cluster);
5  JFreeChart jFreeChart = ChartFactory.createXYLineChart(
6      "Clusterer Result Chart", "Month", "Values",   // y axis label
7      lines,   // data
8      PlotOrientation.VERTICAL,
9      true,  // include legend
10     true,   // tooltips
11     false  // urls
12     );
13 XYPlot plot = jFreeChart.getXYPlot();
14 String label = "Cluster Representative by Average";
15 XYDataset inputDataforLine = converter
16     .convertClusterRepresentativeIntoXYDataset(reresentative, label);
17 plot.setDataset(1,inputDataforLine);
18 XYLineAndShapeRenderer renderer1 = new XYLineAndShapeRenderer();
19 float dash[] = {5.0f};
20 renderer1.setSeriesStroke(
21     0, new BasicStroke(3.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
22     2.0f, dash, 2.0f
23             )
24 );
25 plot.setRenderer(1, renderer1);
26 plot.getRendererForDataset(plot.getDataset(1)).setSeriesPaint(0, Color.black);
27 File file = saveChartAsPNG("jfreetest_allClustersAsChart.png", jFreeChart);
28 return file;
29 }
```

Source Code 6.12: A line chart using JFreeChart

6.5 Evaluation of Clustering Performance

In chapter 5, we described the essential idea of estimating the performance of clustering results. Figure 5.1 in section 5.1 shows an performance evaluation step after each clustering run. We call this process *clustering improvement*. Thereby, an error rate is computed for each clustering run. To this end, we implemented a two-step procedure which is shown in source code 6.13 and 6.14.

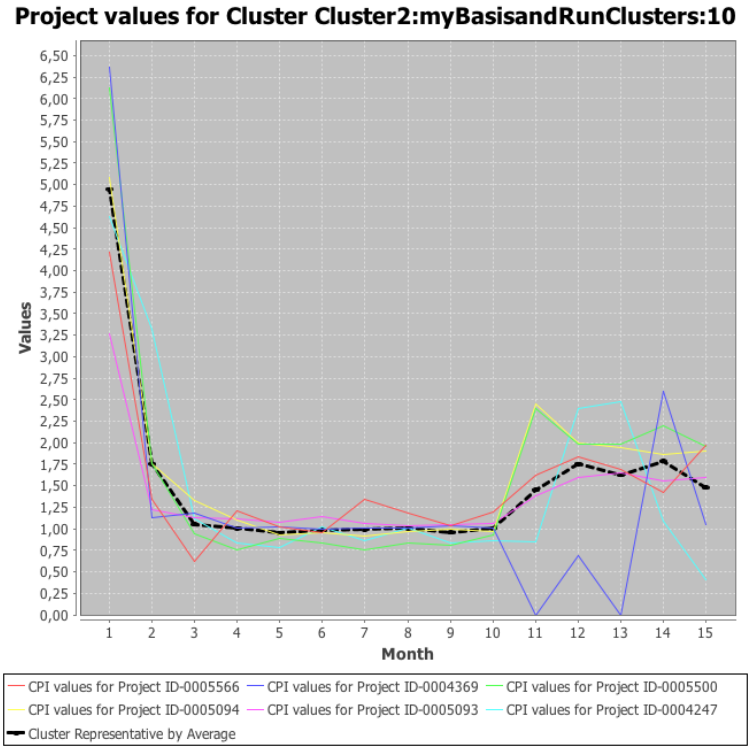

Figure 6.15: Resulting JFreeChart line chart

In the first step, we evaluate the clusterer in the following way: First, we copy all instances of the training set into an *evaluation set* and create an initially empty *test set*. After that, we iteratively move one instance from the evaluation set to the test set (cf. lines 9-10 in source code 6.13). By applying the clusterer using `weka.clusterers.Clusterer.clusterInstance`, the test set instance is assigned to one of the clusters which is memorized (cf. line 12). Continuing, we clear the test set and repeat the process for all other instances in the evaluation set until it is empty. The predictor results are stored in the `PredictionResult` objects that contains the id of the predicted object (`eom`) and the cluster index to which the project has been assigned (cf. lines 16-19).

```
1  List<PredictionResult> result = new ArrayList<PredictionResult>();
2  Instances evaluation_set = DataTransformation
3                  .convertDataVectorListToInstances(vectors, cluster_size);
4  Instances test_set =    new Instances(...);
5  int trainset_size = evaluation_set.numInstances();
6  for (int i=0; i<trainset_size; i++){
7      Instances _temp_evaluation_set = DataTransformation
8                  .convertDataVectorListToInstances(vectors, cluster_size);
9      test_set.delete();
10     test_set.add(_temp_evaluation_set.instance(i));
```

```
11      _temp_evaluation_set.delete(i);
12      Predictor predictor = new Predictor();
13      int cluster_index = predictor.getClusterForPredictionOutput(clusterer,
14              test_set, _temp_evaluation_set);
15      Instance instance = test_set.instance(0);
16      String eom = instance.stringValue(instance.classIndex());
17      PredictionResult predictionResult = new PredictionResult();
18      predictionResult.setCluster_index(cluster_index);
19      predictionResult.setEom(eom);
20      result.add(predictionResult);
21  }
```

Source Code 6.13: Evaluation of Clustering Performance: SetupClusterer

In the second step, we use the result of the first step and compare the values of each project with the values of the cluster representative (cf. source code 6.14). First, for a project, we create the cluster representative of the project's cluster (cf. lines 7-14 in source code 6.14). Then, we compute the average variance of the cluster representative and the project values in percent (cf. lines15-20). This is repeated for each project in the training set. The variances are persisted in the database for each clustering run as a `QualityMeasurement` entity and represent the average error rates of each run (see lines 23-28 in source code 6.14).

```
1  List<Double> run_evaluation = new ArrayList<Double>();
2  List<PredictionResult> predictionResults =
3      new ArrayList<PredictionResult>();
4  predictionResults.addAll(setupClusterer.evaluteClusterer(
5      vectors_list, cluster_size));
6  for(PredictionResult presult : predictionResults){
7      String clusterName = "Cluster" + presult.getCluster_index();
8      Cluster cluster = dataminingFrameworkCrudFacade
9          .getClusterByName(clusterName);
10     DataVector vector = dataminingFrameworkCrudFacade
11         .getDataVectorByEom(presult.getEom());
12     List<Double> cluster_representative = new ArrayList<Double>();
13     cluster_representative.addAll(
14         this.getClusterRepresentativeByAverage(cluster.getName()));
15     EvaluatorByMeanError evaluatorByMeanError =
16         new EvaluatorByMeanError();
17     Double presult_evaluation = evaluatorByMeanError
18         .getMeanErrorForProjectAndClusterRepresentative(
19             cluster_representative, vector);
20     run_evaluation.add(presult_evaluation);
21 }
22
23 for(Double mean : run_evaluation){
24     String qualityValue = mean.toString();
25     String qualityName = ... ;
26     dataminingFrameworkCrudFacade
27         .createQualityMeasurement(
```

```
28                qualityName, qualityValue, clusteringRunName);
29    }
```

Source Code 6.14: Evaluation of Clustering Performance: ClusteringControlleBean

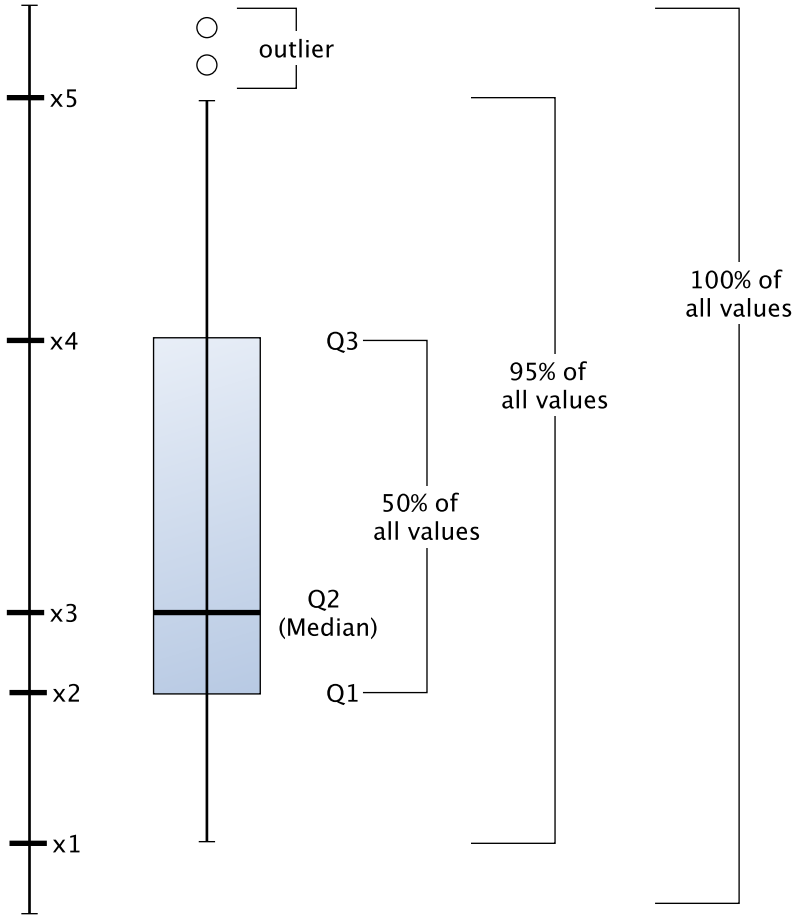

Figure 6.16: Explanation of Box and Whisker Plot

In order to present the results of performance evaluation graphically, we use a box-and-whisker chart. Figure 6.17 demonstrates a resulting clustering performance evaluation in form of box-and-whisker chart created by using the JFreeChart library. Such a visualization enables depicting series of numerical data through their quartiles. Figure 6.16 demonstrates graphical explanation of box-and-whisker plot. The bottom and top of the box are the first (Q_1) and third quartiles (Q_3), and the line inside the box is the second quartile or the *median*. Q_1 implies that 25% of all data values lay under x_2 value and Q_3 corresponds to the 75% of all data values that lay under x_3. The

median represents a value under that lies 50% of all data values.

The ends of the whiskers represents the *upper and lower outlier thresholds* of the data. The upper outlier threshold is computed as $Q_3 + (IQR * 1.5)$, where IQR states for *inter quartile range* that is $Q_3 - Q_1$. The lower outlier threshold is estimated as follows: $Q_1 - (IQR * 1.5)$. The remaining series that are not included between the whiskers, are called *outlier* and typically plotted as dots or small circles. According to the example in figure 6.16 the outlier comprises $2,5\%$ of the whole data set and and $97,5\%$ of all values are equal or less than x_5.

In figure 6.17, each box-and-whisker represents the estimated error rates for a run labeled as `Cluster# <clusters number>`. The box is build according to the `QualityMeasurement` values computed for each clustering run while performing the evaluation procedure. We discuss the interpretation of the obtained evaluation results in chapter 7.

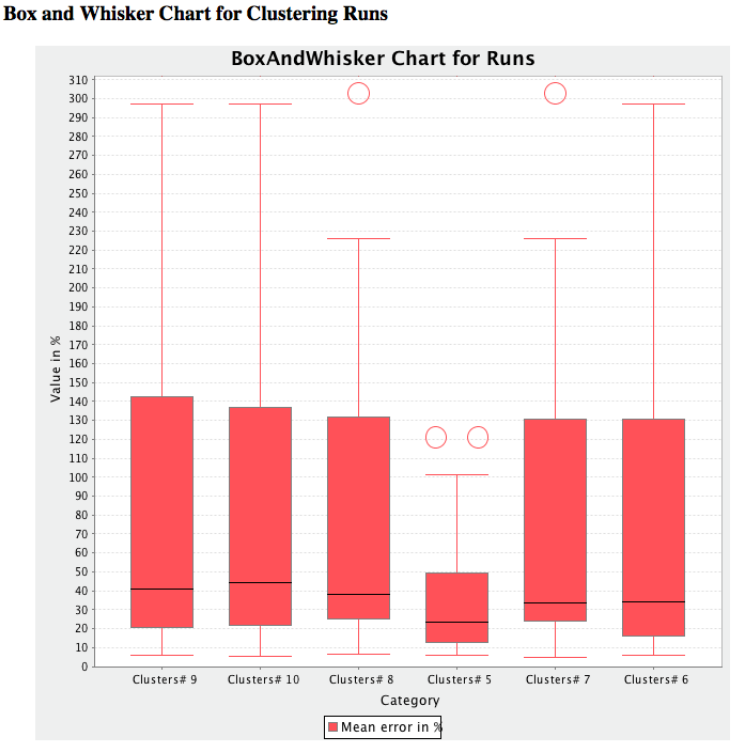

Figure 6.17: Evaluation of Clustering Performance: Box and Whisker chart

7 Evaluation

Contents

7.1 Evaluation of the DataMiningFramework 79
7.1.1 Evaluation Based on Requirements 79
7.1.2 Static Code Analysis 80
7.2 Evaluation of the PredictionTool 82
7.2.1 Experiment: Application Scenario 1 85
7.2.2 Experiment: Application Scenario 2 87
7.2.3 Experiment: Application Scenario 3 92

Based on the requirements defined in chapter 3, we designed the architecture of DataMiningFramework and described it in section 6.2. In sections 6.3 and 6.4, we explained how we integrated different computation and visualization libraries in DataMiningFramework. Subsequently, we evaluate the software prototype's quality and compliance of requirements in section 7.1. Then, in section 7.2, we evaluate the prototype using the implemented PredictionTool through designed experiments with the empirical database of the company ITERGO.

7.1 Evaluation of the DataMiningFramework

In this section, we discuss the evaluation of the DataMiningFramework's quality. For this purpose, we validate the DataMiningFramework according to the functional requirements in section 7.1.1 first. After that, we analyze the code quality using static analysis methods in section 7.1.2.

7.1.1 Evaluation Based on Requirements

In this section, we concentrate on the evaluation according to the functional requirements defined in section 3.2 that we derived from the objectives of this work and stakeholder's needs. According to the Electrical and Electronics Engineers (IEEE) Standard Glossary, a system evaluation during or at the end of the development process serves to *"determine whether it satisfies specified requirements"* [Ele90]. In the following, we discuss the coverage degree of the specified functional requirements:

FR2.1 addresses the requirement of providing loosely-coupled architecture that is based on the independent modules. For this purpose, we designed component- based architecture of the DataMiningFramework as described in section 6.2. The architecture comprises seven components that communicate through *Facades*.

Each component encapsulates a certain functionality based on the concept of *"separation of concerns"* and can be developed independently from the each other. Hence, the requirement **FR2.1** is fulfilled in the realization of the system.

FR2.2 and FR2.3 The requirements define implementation of the data mining steps needed for data mining analysis. Thereby, each step of discovering data knowledge must be encapsulated by a component of the system. We created five components (cf. figure 6.1) for each step needed for the data mining analysis, which are data import, data preprocessing, clustering, classification and prediction and data visualization. Each component implements a set of functions that are sufficient to perform the corresponding step of data mining analysis. Consequently, the system implementation completely covers both requirements.

FR2.4 The requirement prescribes providing extendability and exchangeability of the system. Thereby, the framework realization is required to support the possibility to exchange or extend a component with little effort. For this purpose, we defined several hot-spots that offer extension points of functionally of the DataMiningFramework. The system provides hot-spots for the implementation of different filtering, pre-processing, and clustering algorithms. Thereby, the developer needs to implement a single class to achieve the desired realization of a new algorithm or function, and thus, fulfilling the requirement.

FR2.5 The fifth and the last requirement defines the ability of the software system to be integrated into another system. This implies that the DataMiningFramework must provide a API and a component that enable interaction with an external environment, which is not implemented in theDataMiningFramework by now. Hence, the system architecture must be extended to a component that provides function that connect the external system to functionality of the framework and allows execution of the whole process of data mining analysis without control flow initiated by GUI. However, the effort need to create the integration component is minimal due to the loosely-coupled architecture and the access of each component of the system through facades.

7.1.2 Static Code Analysis

A mature development system's program code should satisfy distinct quality criteria measured by static code analysis. The goal of the static program analysis address not only finding bugs and blockers, it also helps to reveal security and reliability lacks in the system. The static analysis methods detect the errors or code fragments that can result to system's failure in the future. There exists a number of different tools enabling an automatic code analysis. In this work, we used the open-source platform SonarQube [Son] to evaluate the code quality. SonarQube provides various static analysis tools and methods and supports analysis of different program languages.

Figures 7.1, 7.2, and 7.3 show the results of code analysis of the DataMiningFramework (project MA Elena) with SonarQube. The analysis of

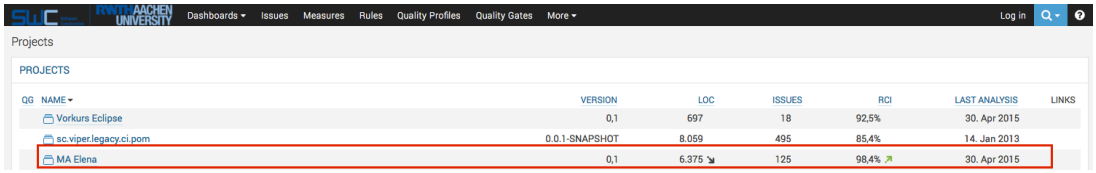

Figure 7.1: SonarQube: Project overview Elena MA

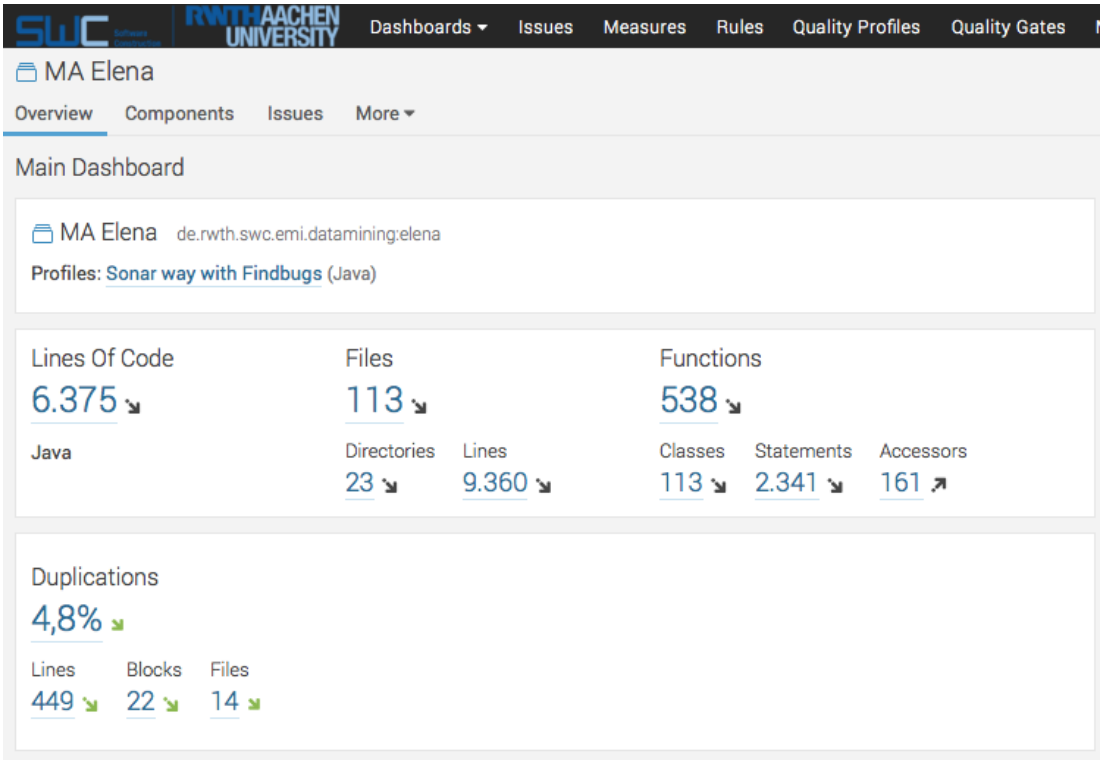

Figure 7.2: SonarQube: Project overview Elena MA

the system's code reveals that the system architecture comprises 6375 lines of code (LOCs), 113 classes, and 538 methods (cf. figure 7.2). The implementation of the DataMiningFramework includes low duplicated code of 4.8%, which is due to some visualization component implementation; e.g., graphical representation in form of different chart types.

The realization of the system has over 98.4% rules compliance, which reflects only few issues and findings in code. In particular, the system has no blocker and critical errors, 32 major issues (preserving stack trace, avoiding duplicate literals, etc.), three minor issues (nested if statements) and 90 info issues (unused modifier) (cf. figure7.3).

The Cyclomatic Complexity (CC) metric used by Sonar (see figure 7.3) indicates the number of linearly independent paths through the source code. The best value for this

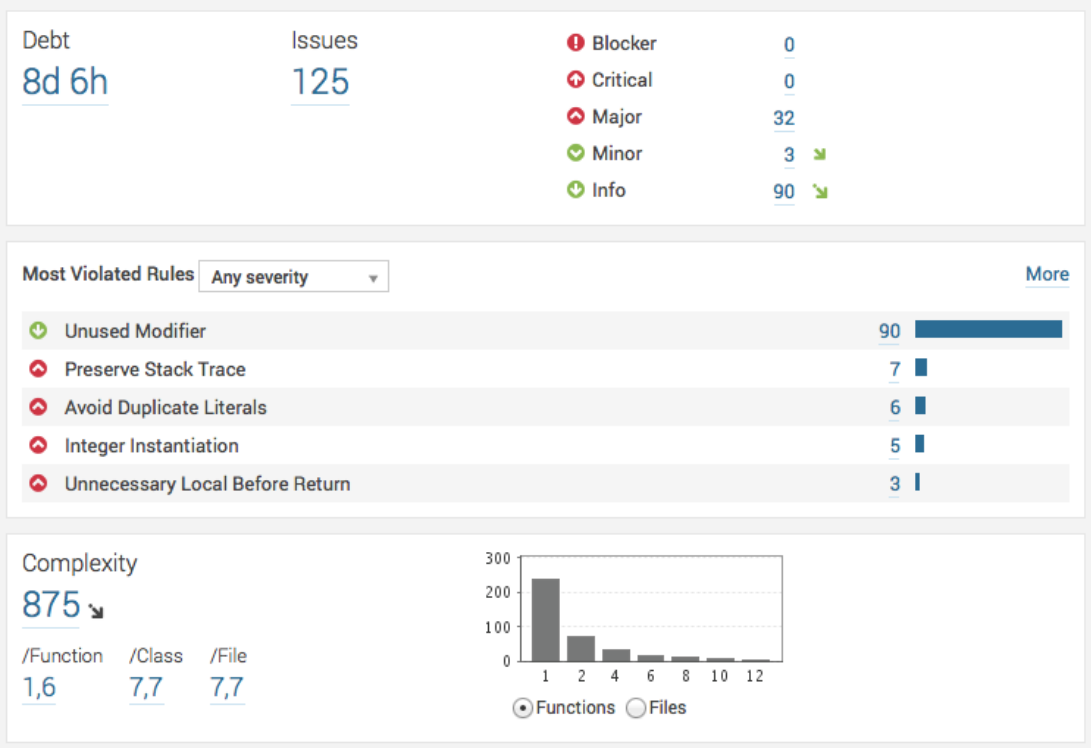

Figure 7.3: SonarQube: Project overview Elena MA

metric is defined by 1, implying that the method is characterized by only one linearly independent path. In general, the complexity per method that is less than 10 refers to the understandable code that enables a convenient testing process. Sonar provides CC measured per class, file, or method. For DataMiningFramework the average complexity is 1.6 per method, 7.7 per class, and 7.7 per file. This indicates that the project provides simple and understandable implementation.

7.2 Evaluation of the PredictionTool

In this section we, discuss the evaluation results of PredictionTool using ITERGO's historical data of 281 projects with CPI and SPI values. Preliminary to the experiments, we list the parameters and the application scenarios for the evaluation.

First, we identify the evaluation parameters: We have to decide on a project durations that are taken into account for clustering. Additionally, we have to specify the vector size, i.e., the number of values for each project that is clustered. Next, the range of the number of cluster k (cf. requirement **FR1.6.1**), i.e., have to define the minimum and maximum for k. Last but not least, the methods for regression and clustering (cf. requirement **FR1.5**). In summary, these are the parameter that must be specified for the evaluation:

- **Project duration range**
- **Vector size**
- **Range of clusters number k**
- **Regression method**
- **Clustering method**

Figure 7.4 depicts a box-and-whisker diagram representing durations of 281 projects that are available for the evaluation. From the diagram, we see that the development duration of all historical projects is between one and 27 months and 50% projects are of four to 12 months duration. This implies that the duration of 25% of the projects is 13 months or greater and the other 25% of projects were developed within one to four months. Note that projects that lasted less than five months are often manageable, and thus, not interesting for prediction. Moreover, we also do not take into account projects that lasted more than 15 months. In other words, to evaluate the PredictionTool, we use the projects of duration five to 15 months. For the experiments, we used CPI values of 151 (53.7%) projects from the empirical database of the company ITERGO.

Choosing the vector size, we must take into account the following issues: On one hand, if we choose a smaller value than the project's number of values, some of these are not considered during the clustering process and we lose information. For example, if we set vector size to 5, we loose 50% of information for projects have ten values, and 75% for projects have 20 values. On the other hand, if the vector size is bigger than the number of values a project has, we must estimate missing values by, for example, using a regression method. For example, for the projects that have ten values and the vector size is 20, the amount of the estimated values comprises 50%, accordingly for five values we have to estimate 75% of values. The is to say, if we choose a too high value, we would mostly use estimated values for clustering, and thus, affecting the accuracy of the prediction.

Due to time constraints of this thesis, we commit to fix values for the last three parameters for all experiments. This commitment is based on some preliminary observations during tests. In particular, we define the range for the number of clusters to be five to ten, i.e., $k \in [5; 10]$ clusters. To compute missing values, we use a linear regression function. Last, we use the k-Means clustering algorithm to evaluate our PredictionTool.

During the experiment, we identified a technical restriction by means of processing time that is needed to calculate clustering runs for different k and their performance evaluation. All experiments were performed on Mac OS X, Version 10.9.5 with Processor 2 GHz Intel Core i7. For a data set comprising up to 50 projects, the calculation terminated in less than one minute. For a data set that comprises 50 to 100 projects, the time needed to perform and evaluate the clustering process risen to up to five minutes. While trying to test a data set of 150 projects, we had to abort the process after 30 minutes without its termination. A reason for a long processing time is a huge number

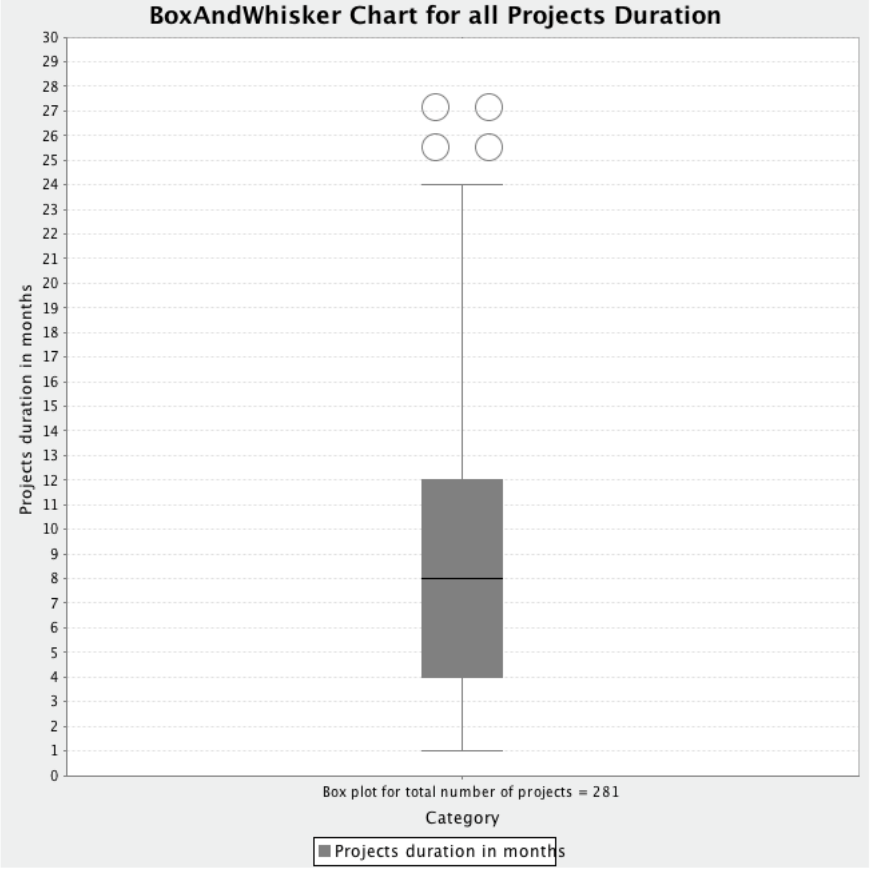

Figure 7.4: Resulting PrimeFaces line chart

of read and write operations in the databases needed to perform the complete clustering process and its performance evaluation. In this work, we used MySQL database to store and retrieve the data. A possible solution for reducing of computation time might be replacing SQL database by a NoSQL database, such as document or graph databases, that might be more suitable for the type of data structures used in this work, i.e. data vectors.

The latter issue explains why we restricted the number of projects per each defined group to maximum 100 projects during evaluation experiments.

We designed the following possible application scenarios of PredictionTool:

Application Scenario 1 The first scenario targets to estimate data quality. Hence, we evaluate the clustering model w.r.t. different data sets that we built from different project duration range. The results are presented by using mean error that we computed from the difference of each project compared to the representative of cluster to which the underline project is assigned (cf. section 6.5).

Application Scenario 2 The objective of the second scenario is to analyze the variance of

mean error. The variance occurs due to the random estimation of missing values by using regression. We compute the mean error variance by performing *ten* iterations of clustering run for the same number of clusters k for each $k \in [5, 10]$ and calculate the average variance of error based on all iterations.

Application Scenario 3 In the last scenario, we evaluate predictive facilities of the PredictionTool trough step-wise prediction of a given project. We extracted one project from the evaluation data and use it to estimate prediction error. To this end, we enter the first five values of the extracted project into PredictionTool which results in a prediction of a the value for the sixth month. With this value, we then predict the value for the seventh month and so on. Finally, we compare the predicted points with the original values of the project to determine the prediction error.

In the next three sections, we present the statistical results of the three experiments based on the application scenarios and discuss their interpretation.

7.2.1 Experiment: Application Scenario 1

In this section, we analyze the results of the first scenario by applying linear regression and k-Means clustering algorithms to different data sets. We divided the evaluation data into groups that contain more than ten projects and at max. 100 projects (due to performance reasons). To this end, we considered groups were all projects have the same duration and groups with a defined duration range.

Further, we defined the vector size for each group as the upper bound value of each interval. This implies that for projects with duration less than this value (vector size) the missing values are calculated by using regression. Taking this into account, we chose the groups with a duration range w.r.t. how many missing values are needed to be estimated: few, medium, many.

Table 7.1 shows the evaluation results for each group ordered by $Q_{2_{min}}$ ascending. The first column lists the group names, where the number represents the duration of the projects; e.g., **B 10-15** is the group with all projects of duration ten to 15 months. Further, the table provides columns with group and vector size, i.e., the number of projects in each group and the number of points of each project used for the clustering, respectively. The next two columns present the maximum and the minimum mean errors of the clustering performance, and the minimum and maximum of the first quintile (Q_1) and the third quintile (Q_3), respectively. The sixth column shows the interval of the median for each group. In the last column, we listed the best number of clusters k by means of the lowest value of the median identified for the group.

Discusson

Form table 7.1, we derive that the range of all error values is distributed between 0% to 420%. However, the average value of all medians is approximately 63%, which imply that 50% of all errors lie under 63%. The best results where indicated for the group

Group	Group Size	Vector Size	Mean errors	$[Q_1; Q_3]$	$[Q_{2_{min}}; Q_{2_{max}}]$	Best k
B 14-15	13	15	[3%; 76%]	[8%; 43%]	[12%; 27%]	8
B 10	16	10	[4%; 90%]	[13%; 53%]	[25%; 32%]	5
B 13-15	18	15	[0%; 150%]	[15%; 100%]	[35%; 90%]	7
B 5-10	104	10	[3%; 250%]	[32%; 140%]	[45%; 64%]	5
B 10-15	61	15	[10%; 240%]	[25%; 140%]	[45%; 110%]	10
B 6-8	63	8	[6%; 200%]	[26%; 108%]	[49%; 69%]	5
B 8-10	51	10	[0%; 310%]	[26%; 159%]	[50%; 102%]	7
B 6	19	6	[0%; 200%]	[14%; 108%]	[52%; 74%]	10
B 8	22	8	[0%; 420%]	[40%; 270%]	[62%; 167%]	9
B 8-15	100	15	[5%; 415%]	[30%; 280%]	[65%; 105%]	9

Table 7.1: Summary of Experiment 1

B 14-15. Figure 7.5 shows the box-and-whisker plot for that group and is generated from the mean errors of cluster representatives by clustering projects. The range of all error values is distributed form 3% to 76%. The maximum and the minimum of the first and the third quintile disseminates between 8% and 43%. The median range of values is indicated between 12% and 27%. The best result for this group is identified for number of clusters $k = 8$ where 50% of all errors lie under 12%.

The second best group w.r.t. low mean error is the group **B 10**. Both, group **B 14-15** and group **B 10**, have similar characteristics: they includes projects with no or only few missing values. Meaning that the regression to estimate missing values is applied to only few values or is not at all. One might conclude that the application of the regression to cover missing value gaps causes interferences in the prediction result. However, the assessment results for the other two group with no need to calculate missing values, namely, group **B 6** and **B 8**, show significantly higher error rates (cf. table 7.1). In fact, group **B 8** has the highest error rate.

Figure 7.6 illustrates graphical representation of group **B 8** in form of box-and-whisker plot. The maximum error rate for this group is 430%, which is the highest value in this experiment, and the mean of median of error rate for all evaluated numbers of clusters k is 114.5%.

Since we use k-Mean algorithm, which is based on a distance function, to cluster the data, the reason for the high error rates in groups **B 6** and **B 8** can be explained by the low denseness of the projects within the groups. Figure 7.7 shows two of the clusters, cluster 1 and 6, for group **B 8** where $k = 9$. The two clusters are the only clusters that include more than two projects for this clustering. Both clusters are low-dense with respect to the distance between projects that are assigned to the corresponding cluster.

Figures 7.8 and 7.9 illustrate the comparison of the projects' density for groups **B 6**, **B 8**, **B 10** and **B 14-15**. Taking a closer look, we can see a similarity of the groups

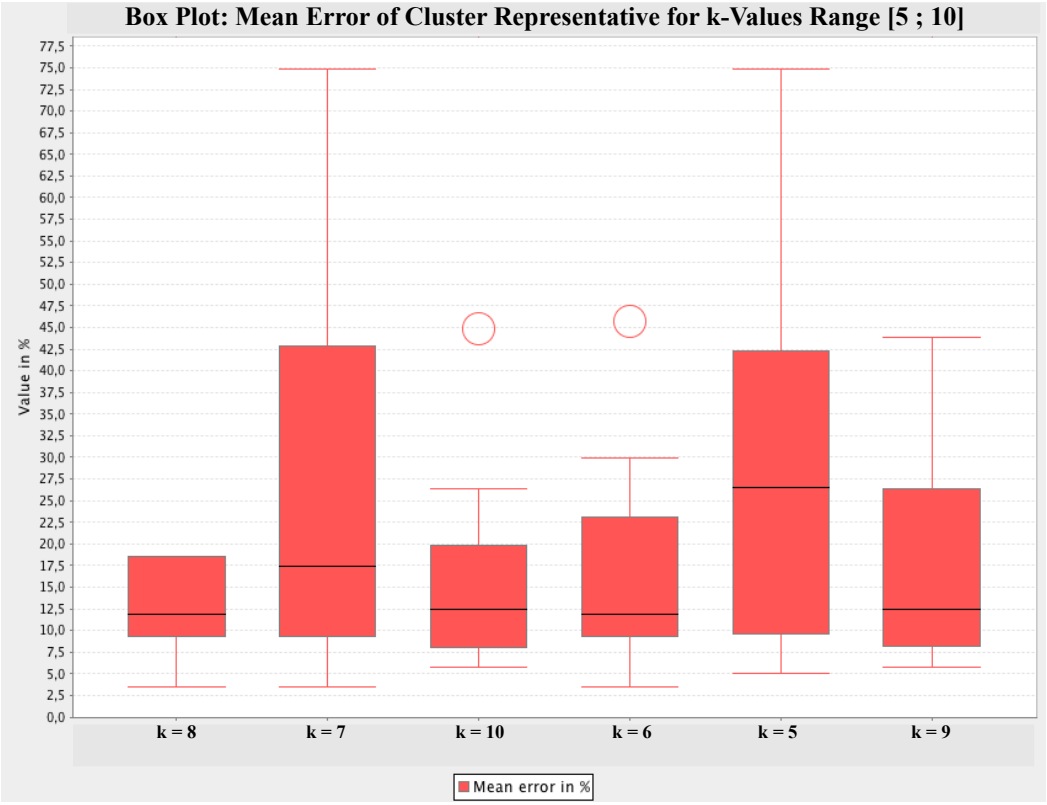

Figure 7.5: *Application Scenario 1* for data set **B 14-15**

B 6 and **B 8**: both contain projects where the CPI value trends are of low denseness. In contrast, the CPI values of the projects in groups **B 10** and **B 14-15** proceed in a small interval $[1.0; 1.5]$ (high denseness) after 30%-50% w.r.t. the duration of projects development time.

The results of this analysis, based on the the evaluation of clusterer model (cf. section 6.5), indicate that neither the computation of missing values, nor the group size seem to matter for the performance of the clusterer (cf. table 7.1). However, we saw that groups, for which clusterer shows low error rates, and groups, for which clustering demonstrates significantly higher error rates, possess in each case similar characteristics. Especially, the more similar projects w.r.t. their CPI values a data set has, the better the clusterer performs.

7.2.2 Experiment: Application Scenario 2

For a given vector size of a clustering run, some projects might have less values. Thus, this missing values need to be estimated. To this end, we apply a regression method

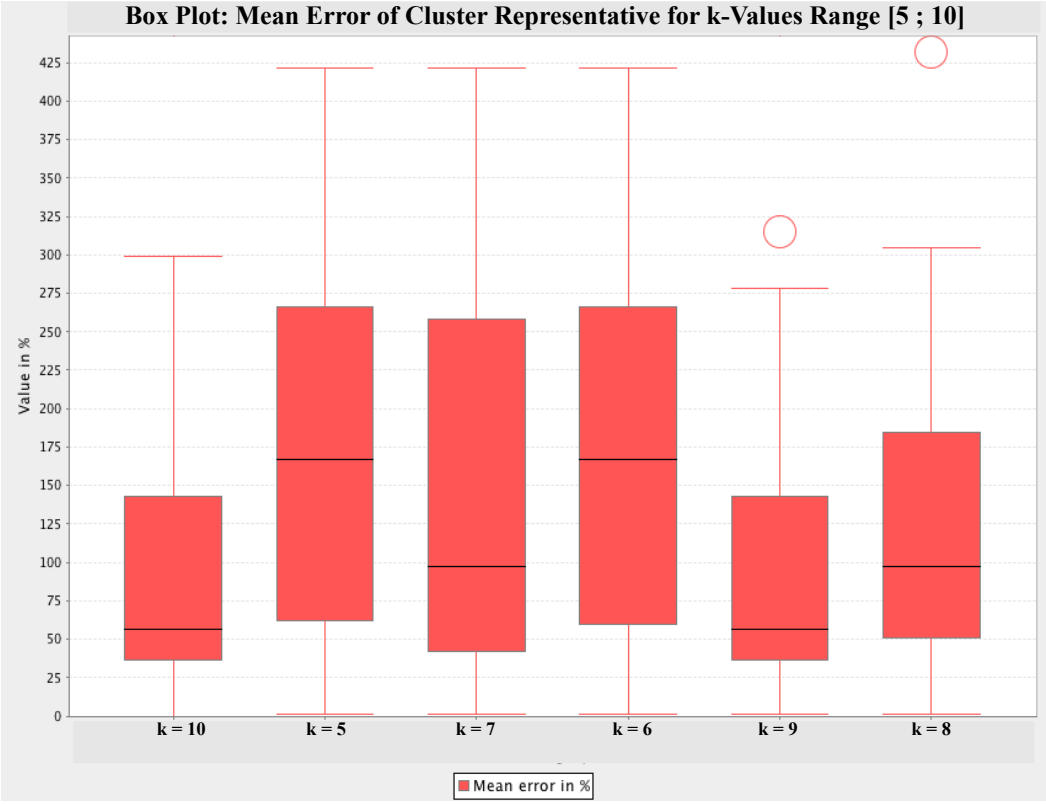

Figure 7.6: *Application Scenario 1* for data set **B 8**

on the project vectors. The algorithm that we implemented in this thesis works in the following way: it computes a regression line for the given values of the project vector, and then, randomly pics values from the regression line for the missing values. Due to the randomness of generated values, the error rates of clustering vary within the certain interval.

This section describes the second application scenario that aims to determine how big is the error variance for each number of clusters $k \in [5; 10]$. To this end, we performed ten iterations of clustering the same data set for each k and computed the average variance of error rate. We compare the computed variance for different group of data sets, which are:

B 14-15 contains 13 projects with duration range between 14 and 15 months.

B 10-15 The group includes 61 project with the duration range between ten and 15 months.

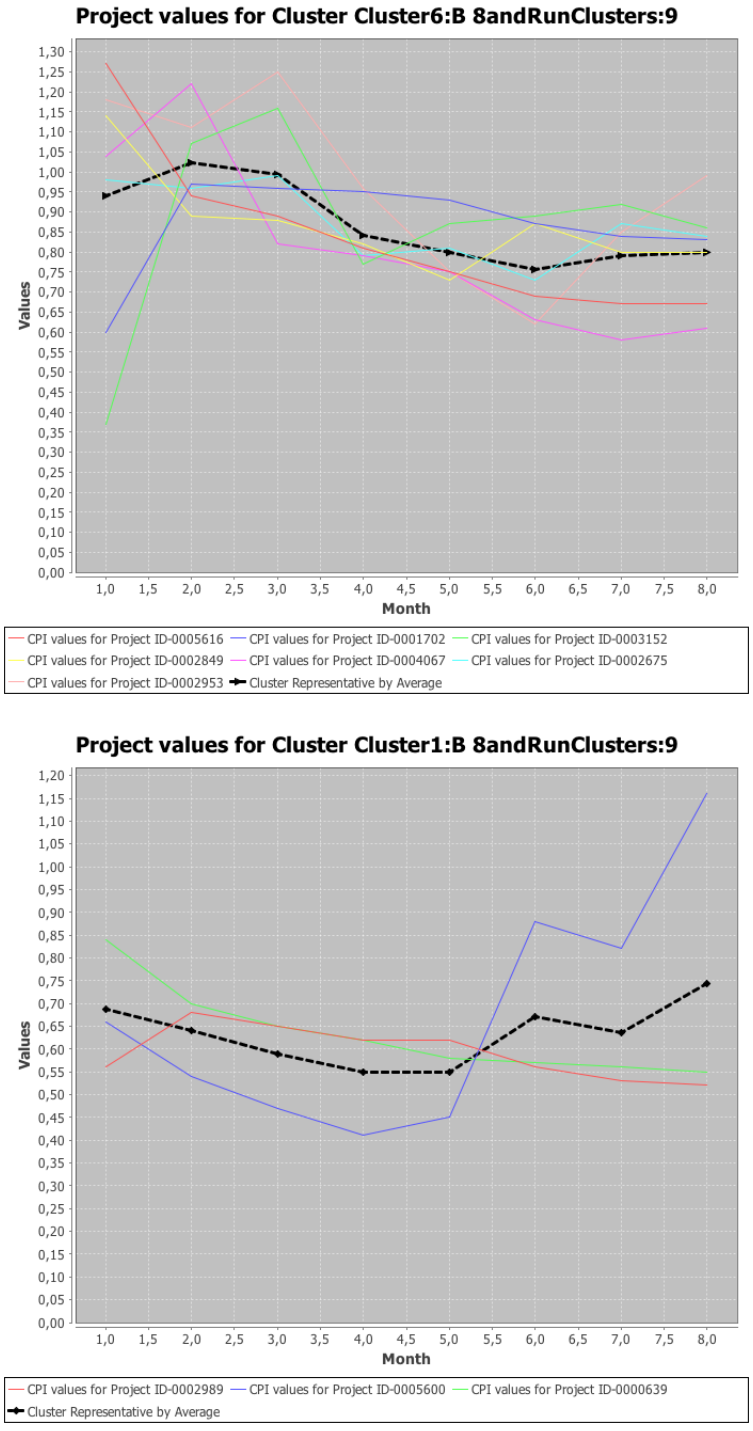

Figure 7.7: *Application Scenario 1* Resulting clusters for group **B 8** ($k = 9$)

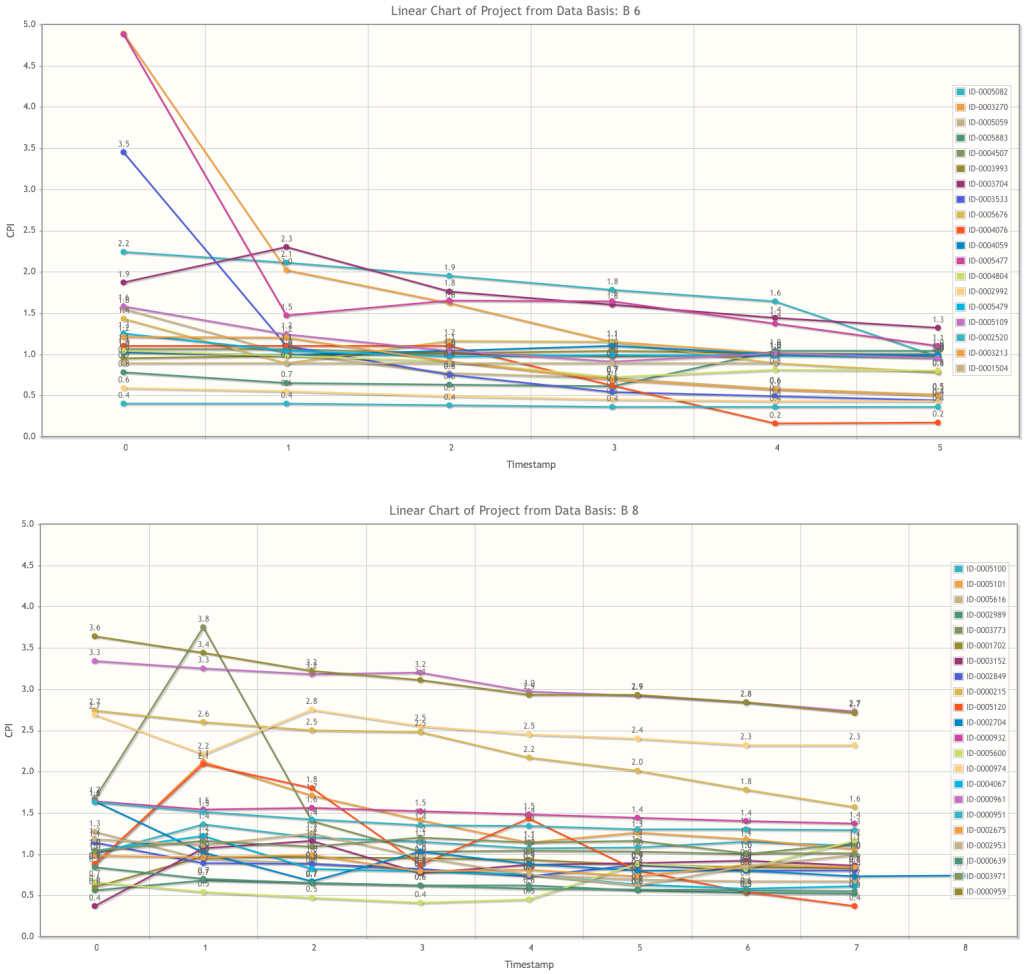

Figure 7.8: *Application Scenario 1* Projects overview for groups **B 6** and **B 8**

B 5-10 This group of data contains 104 projects with duration range between five and ten months.

B 8 includes 22 projects with duration eight months.

Table 7.2 shows the results of the second experiment. Columns two to seven present the average variance of error rate for each group in percent for each $k \in [5; 10]$, respectively. The second last column summarizes the lowest and the highest error variances and the last column states the average error variance for each group. The forelast and last rows present the same for each number k of clusters.

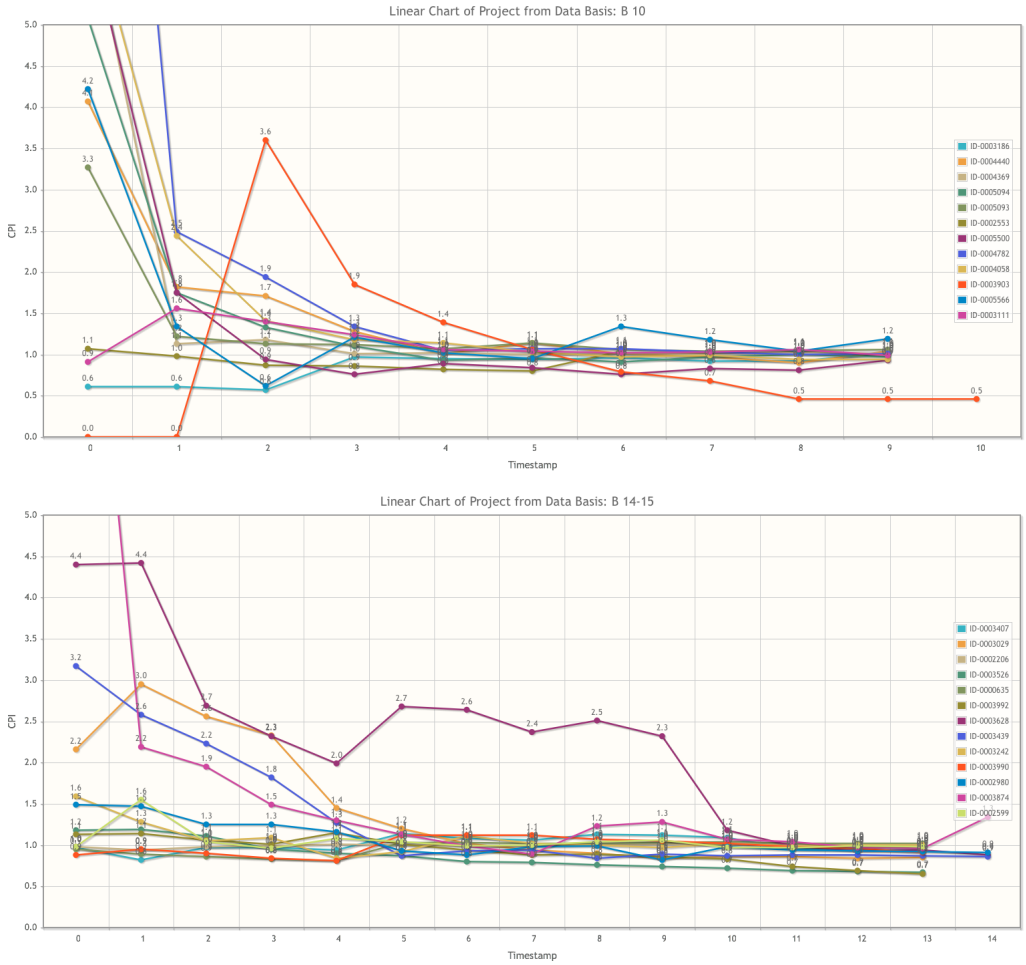

Figure 7.9: *Application Scenario 1* Projects overview for groups **B 10** and **B 14-15**

Discusson

According to the results of the second application scenario that are presented in table 7.2, the values of error variance are spread between 11% and 41%. The difference between average variance values between groups is in between 26% and 30%. Group **B 14-15** has the highest average error variance and groups **B 10-15** and **B 8** the lowest. The highest average variance value w.r.t. number of clusters k is for $k = 6$, 33%, and the lowest value of 21.5% for $k = 7$. The average value of performance error variance for groups and number of clusters k is about 27%. This analysis indicates that the error variance due to the randomness of the missing value computation, is negligible being about 27%.

Group	Mean (%)						$[min\%; max\%]$	$Av.$ (%)
	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$		
B 10-15	18	41	26	19	31	21	[18; 41]	26
B 14-15	37	33	25	30	34	20	[25; 37]	30
B 5-10	35	25	11	28	30	32	[11; 35]	27
B 8	24	32	24	18	29	27	[18; 32]	26
$[min\%; max\%]$	[18; 35]	[25; 41]	[11; 26]	[18; 30]	[29; 34]	[20; 32]		
$Average(\%)$	28, 5	33	21, 5	24	31	25		

Table 7.2: Summary of Experiment 2

7.2.3 Experiment: Application Scenario 3

The objective of the third experiment is to evaluate the predictive facilities of the PredictionTool. Within this experiment, we simulate the prediction process by applying PredictionTool to a given project. In order to evaluate predictor performance, we extracted one project form the available data set and trimmed 70% of its values. The remaining 30% of values we used as prediction basis, i.e., the start points for prediction. We, then, compared the predicted with the original values.

For this experiment we picked a 15-months project as the test project. Figure 7.10 shows the values of this project. We use the first five values from month one to five as a start, and then, applied PredictionTool to predict the values for the months six to 15.

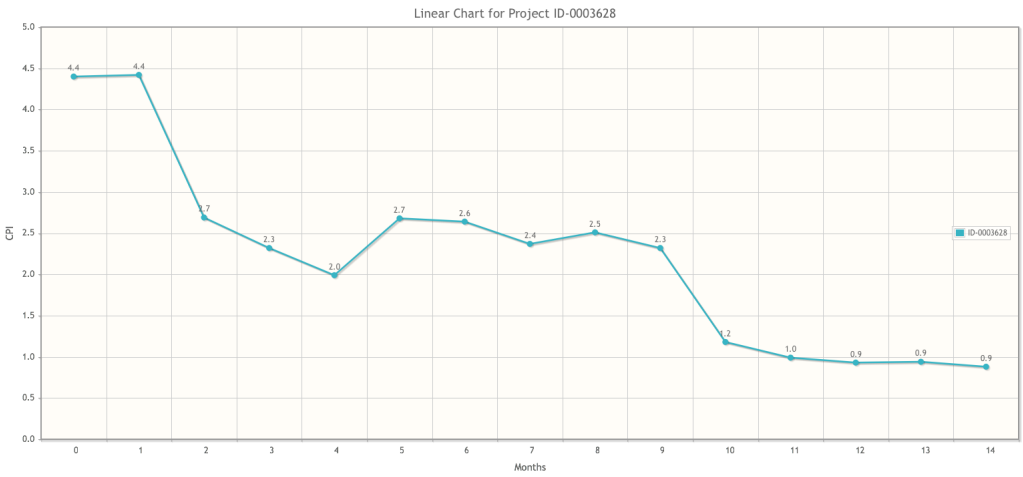

Figure 7.10: Experiment 3: Test project

In order to compare different scenarios for prediction, we split the experiment into three sub-scenarios: *70%-Prediction*, *35%-Prediction*, *Step-wise-Prediction*. In former, *70%-Prediction* scenario, the PredictionTool generates prediction values for interval $[6; 15]$. In the second scenario, the *35%-Prediction*, we split the prediction interval into two sub-intervals $[6; 10]$ and $[11; 15]$, where the predicted values for $[6; 10]$ are used with $[1; 5]$ to predict values for $[11; 15]$. Third scenario, the *Step-wise-Prediction*, we applied the PredictionTool step-wise to forecast the next unknown value, and then, adding it to the prediction basis using it in the subsequent step.

Experiment 3.1: 70%-Prediction

For this experiment, we used the following groups of data:

B 8-15 contains 100 projects with duration range between eight and 15 months.

B 10-15 The group includes 61 project with the duration range between ten and 15 months.

B 14-15 This group of data contains 13 projects with duration range between 14 and 15 months.

Figures 7.11, 7.12, and 7.13 reflect the outcome of prediction for three data sets. On the left-hand side, the chart shows the cluster, comprising the projects and the cluster representative, that is used for the prediction. On the right-hand side, the chart shows the prediction of the project values. Here, the solid (blue) line represents the values of the original test project from above (project *ID-0003628*), the dashed (black) curve reflects the values of the cluster representative; in other words, the prediction line, and the dashed (green) curves represent the maximum and the minimum values of the corresponding cluster.

The mean errors for **B 8-15**, **B 10-15**, and **B 14-15** using *70%-Prediction* are 114.12%, 50.51%, and 31.83%, respectively. That is to say that the best result of prediction is for group **B 14-15**. However, the prediction for this group results in high errors for the first half of the prediction interval $[6; 10]$ and is compensated by the second half $[11; 15]$. Moreover, taking a closer loot at the trends, the predictions forecast values of high error rate either for the first half or the second half of the 70%. While this may be true, the first five values are descending, and thus, the sudden ascend is in fact hard to predict.

Experiment 3.2: 35%-Prediction

In order to evaluate the *35%-Prediction* scenario, we used two groups of data:

B 8-10 contains 51 projects with duration range between 8 and 10 months.

B 13-15 The group includes 18 projects with duration range between 13 and 15 months.

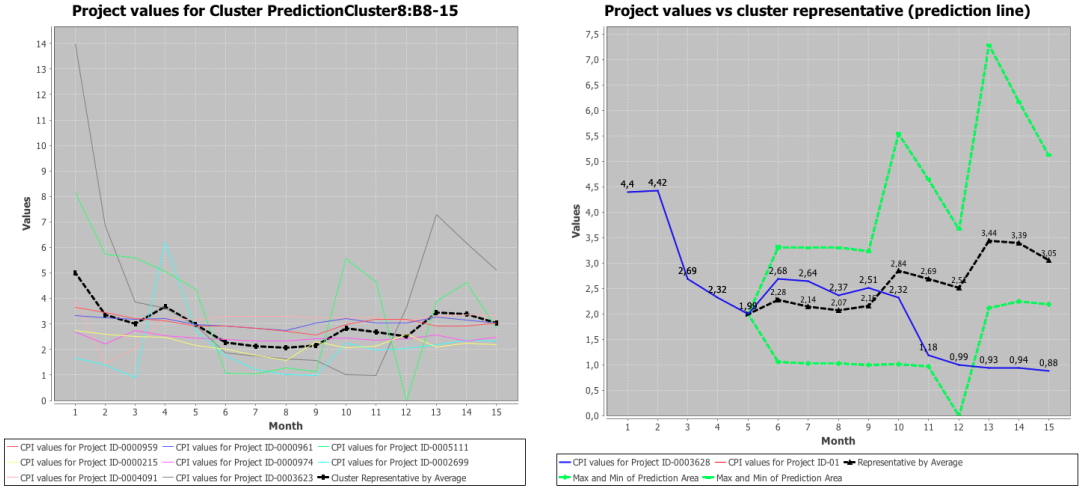

Figure 7.11: Experiment 3.1: Prediction results for 70%-Prediction for data set B 8-15

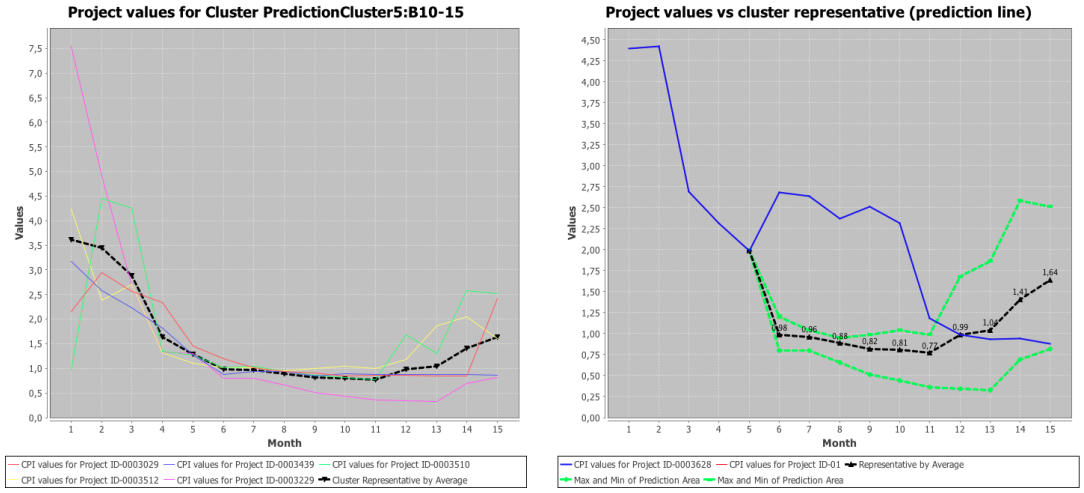

Figure 7.12: Experiment 3.1: Prediction results for 70%-Prediction for data set B 10-15

We used the first data set **B 14-15** for the forecasting of the first prediction interval and the second data set [B 13-15] to predict the second interval.

Figure 7.15 shows the resulting chart of the prediction. As before, the blue curve represented the original project progress. The dashed black curve is the cluster representative with the corresponding maximum and minimum bounds depicted by the green dashed curves. The red line reflects the predicted values for the test project (in the legend *ID-01*) that were generated in the previous prediction step. The red line and the cluster representative assemble the set of predicted values for the test project. The mean error of the *35%-Prediction* scenario results to 55.59%. Similar to the experiment 3.1, the prediction line of the *35%-Prediction* scenario does not detect the curve fluctuation.

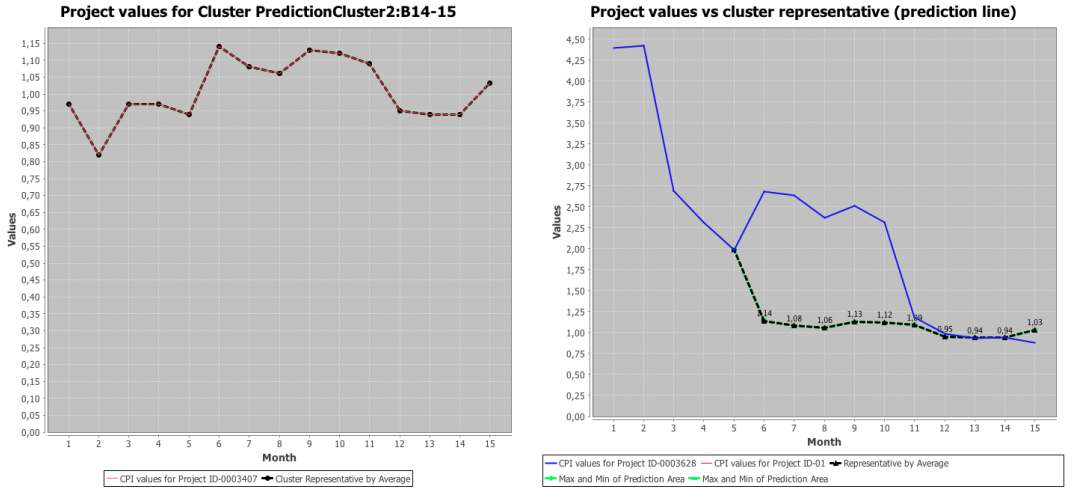

Figure 7.13: Experiment 3.1: Prediction results for 70%-Prediction for data set B 14-15

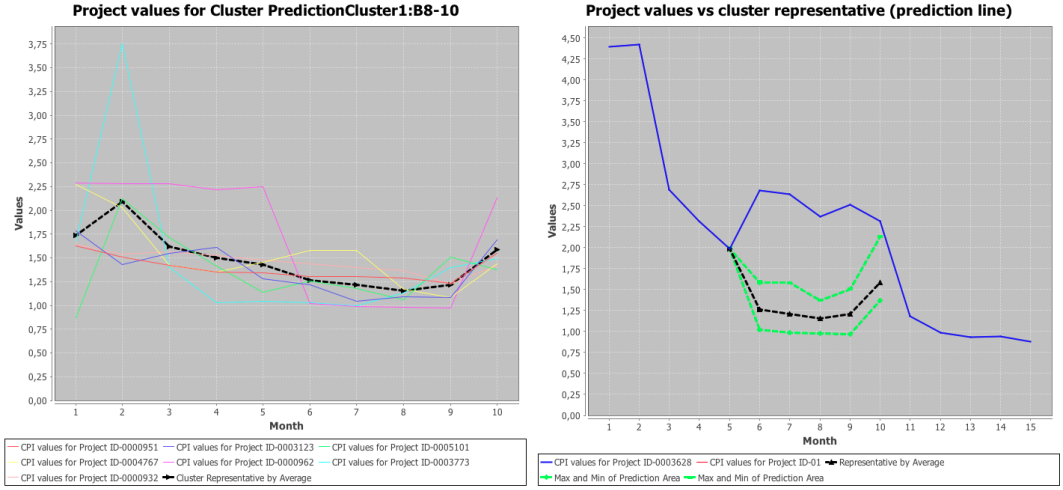

Figure 7.14: Experiment 3.2: Prediction results for 35%-Prediction: Interval $[6; 10]$

Experiment 3.3: Step-wise-Prediction

In this experiment, we performed step-wise prediction by applying the PredictionTool to only two and three months ahead per each step. In the successive step, the predicted values are then used to predict the next period's values. The prediction of each step is based on the results obtained from the previous step. Fro this experiment we split the prediction into fore intervals: $[6; 8]$, $[9; 10]$, $[11; 12]$, $[13; 15]$ and used groups **B 6-8**, **B 8-10**, **B 10-12**, and **B 12-15** respectively. The groups are defined as follows:

B 6-8 contains 63 projects with duration range between six and eight months.

B 8-10 The group includes 51 projects with duration range between eight and ten

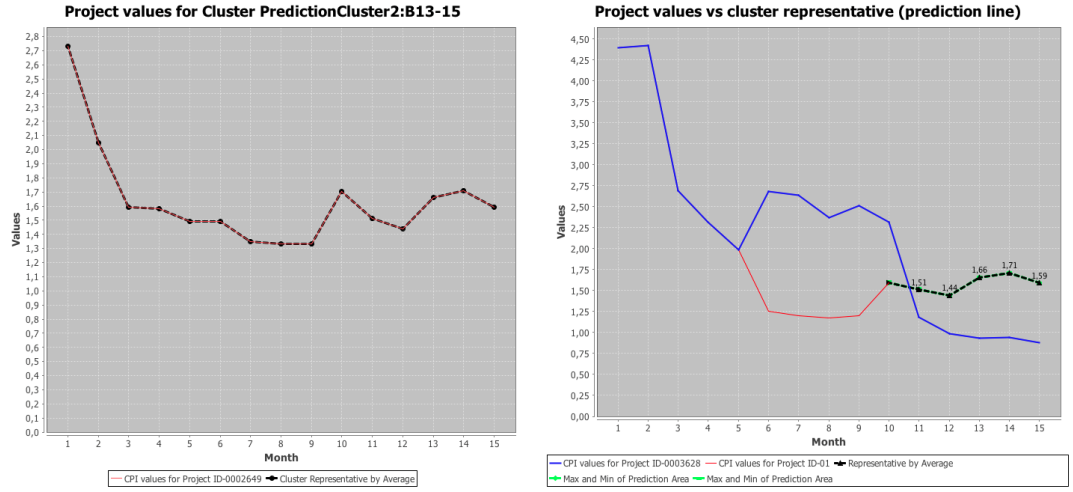

Figure 7.15: Experiment 3.2: Prediction results for 35%-Prediction: Interval [11; 15]

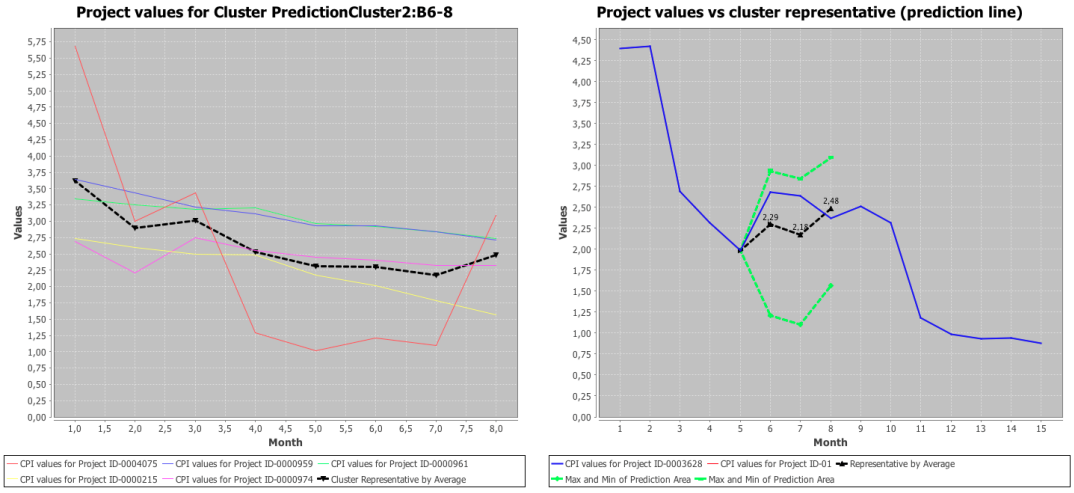

Figure 7.16: Experiment 3.3: Prediction results for 35%-Prediction: Interval [6; 8]

months.

B 10-12 The group includes 43 projects with duration range between ten and twelve months.

B 12-15 The group includes 33 projects with duration range between twelve and 15 months.

The computed mean error for this scenario is 37.58%. The graphical representation of the results for each prediction interval is presented in figures 7.16, 7.17, 7.18, and 7.19, respectively. The obtained results show that the *Step-wise-Prediction* scenario

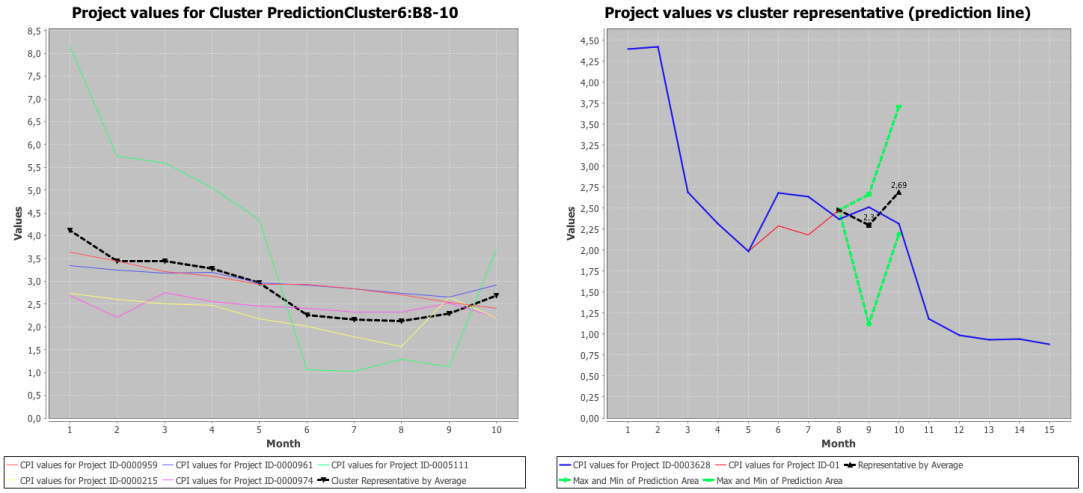

Figure 7.17: Experiment 3.3: Prediction results for 35%-Prediction: Interval $[9; 10]$

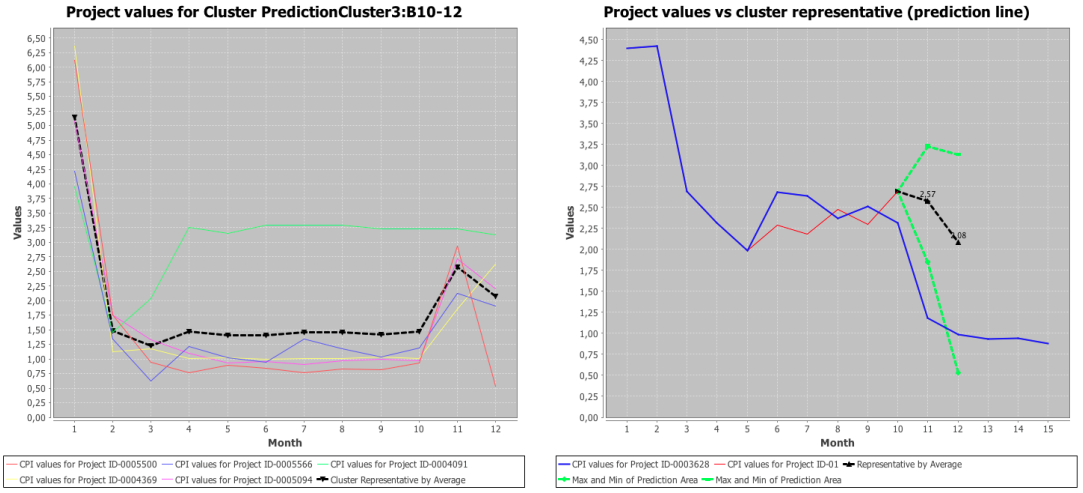

Figure 7.18: Experiment 3.3: Prediction results for 35%-Prediction: Interval $[11; 12]$

outperforms the *35%-Prediction* and the *70%-Prediction*. Additionally, the prediction line of the step-wise prediction shows better results in means of slope and decline in comparison to the original project's progress.

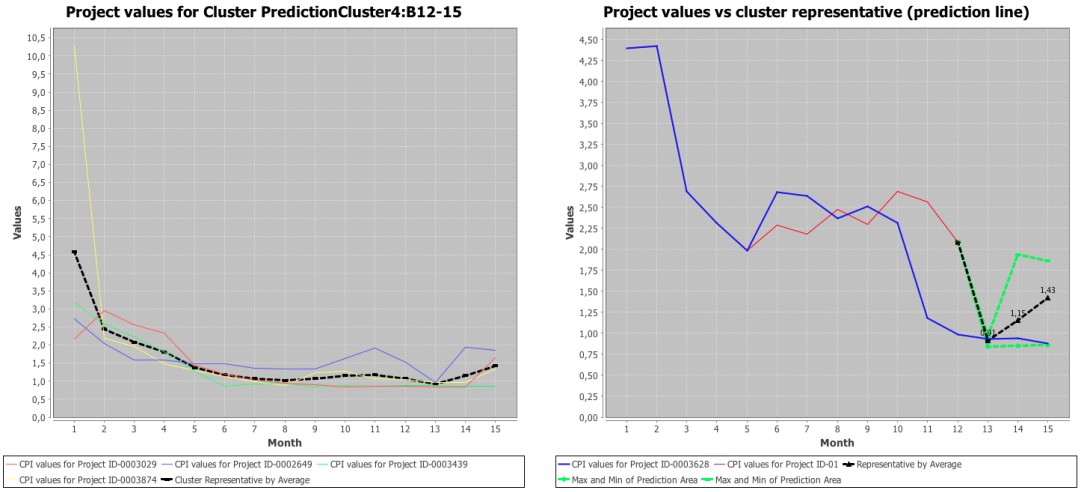

Figure 7.19: Experiment 3.3: Prediction results for 35%-Prediction: Interval $[13; 15]$

8 Conclusion

Contents

8.1 Summary 99
8.2 Future Work 100

In this chapter, we conclude our work. Having a concept and an implementation for our framework and the prediction tool, we would like to summarize our work in section 8.1. Moreover, we present open questions for future research work and discuss possible improvements of the DataMiningFramework and PredictionTool in section 8.2.

8.1 Summary

This master thesis contributes to the designing of a data mining framework, DataMiningFramework, for automatic prediction using data mining methods on the historical data in specific problem domains. Specifically, we use unsupervised learning algorithms in order to classify the data. Furthermore, we implemented the necessary steps required for knowledge discovery from data that include data import, data preprocessing, clustering, prediction, and data representation. Thereby, we encapsulated each step in a component of the DataMiningFramework and implemented several hot-spots for the extendability and changeability. To show how and that the DataMiningFramework work, we created the software prototype PredictionTool and evaluated it by using the empirical database of the company ITERGO.

In context of this work, in the first place, we performed a comprehensive literature research of existing related studies to identify the state-of-the-art in chapter 4. The literature review showed that there exist a number of studies addressing project prediction methods. However, we concluded that non of the reviewed researches describe techniques for predicting CPI and SPI values of a project. Furthermore, we pointed out that most approaches do not provide reliable results and give little information about the tools applied.

According to the objectives for this work considered in chapter 1, we identified stakeholders and engineered requirements in chapter 3. Thereby, we separated the requirements formulation into two groups: requirements for the PredictionTool and the requirements addressing the DataMiningFramework. Based on the derived requirements, we considered the basic concept of the future software for automatic prediction in chapter 5. The approach is based on discovering similarity patterns in historical data and and generating a classifier. Then, with this classifier, an ongoing

project is assigned to a data pattern class and its future values are predicted using that class' representative.

Next, we designedthe domain model and the architecture of the DataMiningFramework in chapter 6. Based on the design principles of software framework construction, we designed a component-based architecture for DataMiningFramework. Thereby, each component is constructed according to its own area of concern or functionality and provides a business interface through Facades. Besides the components implementing the required steps of knowledge discovery from data, the DataMiningFramework provides visualization component for graphical representation of clustering and prediction outcomes.

In chapter 7, we evaluated the DataMiningFramework according to the engineered requirements. The evaluation showed that all requirements but the one are fulfilled by the implementation of the DataMiningFramework. The last requirement addresses the integration of the DataMiningFramework into another system and is a future improvement issue.

Additionally, to validate the DataMiningFramework, we implemented the PredictionTool that uses the major functions of the DataMiningFramework to forecast a project's progress in the future. In order to evaluate the predictive performance of the PredictionTool, we deliberated three application scenarios. In chapter 7, we present the obtained results for each scenario in statistical and graphical form. The results indicated that the tool performs best when the projects inside the prediction class are close to each other and the prediction is limited to only a few time units ahead.

8.2 Future Work

The outcome of this master thesis can serve as a basis for further research work. In this chapter, we give an overview of open questions and improvements in the context of automatic forecasting in specific problem domains that was untouched in this thesis and can be addressed in the following works. For this purpose, we separated this section into two parts. The first part includes challenges and improvement suggestion for the DataMiningFramework architecture, and, in the second part, we describe further possible experiments that can be performed to explore other parameters of clustering to improve the outcome of prediction.

DataMiningFramework

The proposed architecture of the DataMiningFramework can be extended by an API and a component that enable interaction with an external environment. This will make the integration of the DataMiningFramework into a measurement system possible. Consequently, the DataMiningFramework might become a module that is used for different data mining tasks of a larger system.

We implemented only one possibility for data import in the DataMiningFramework, namely the import from an Excel file. Hence, the

new import functions that support, e.g., import from SQL databases, can be added to the set of DataMiningFramework's functions. Also, the import task might be replaced in the future by a measurement system. Therefore, the implementation of the framework must be adapted to offer this possibility.

We implemented the component for the presentation function (UI) of the system using the JSF framework. JSF is a server-side framework which implies that all tasks needed to build the UI is done on the server. The process of streaming them back to the application produces high network load. Due to the server-side characteristics of the JSF framework, we need a lot of server memory to create an application which negatively influences the system scalability. Hence, the JSF framework can be replaced by a newer framework, e.g., Angular JS, which provides client-side implementation and is based on the single page application model.

A further issue addresses enhancement of the level of system's performance. In section 7.2, we briefly described efficiency limitations of the computation core of the system. We concluded that the processing time might be reduced by replacing SQL database by a NoSQL database, which implies that the database API must be customized depending on the used database type.

Last but not least, the hot-spots provided by the architecture can be used to create new algorithms for data selection, computation of missing values in time series, and new clustering methods. Also, other data mining and graphical tools and libraries can be integrated into the DataMiningFramework.

PredictionTool

A huge challenge for the success of the prediction approach is to find out the best parameters values and their combinations. Hence, the PredictionTool can be used to estimate the best parameters values. Due to the time constraints of this thesis, we did not evaluate the clustering performance for an application scenario when the *vector size* parameter is not equal to maximum project duration. In this case, not entire project duration is used to perform clustering. Also, we did not evaluate runs for $k \leq 4$ and $k \geq 11$.

For our experiments, we applied the k-Means clustering algorithm and used linear regression to compute the gaps in time series and missing values when the vector size was grater than project duration. Hence, further clustering and regression methods can be applied and evaluated. Additionally, a further application scenarios for prediction process can be considered, in order to determine the best fitting approach.

The DataMiningFramework is designed to be applied for different problem domains that do not necessary deal with project management. Hence, other tools might be implemented using the main components of the DataMiningFramework to estimate future values in a different domain.

9 Appendix

9.1 Experiment 1: Box Plot Diagrams

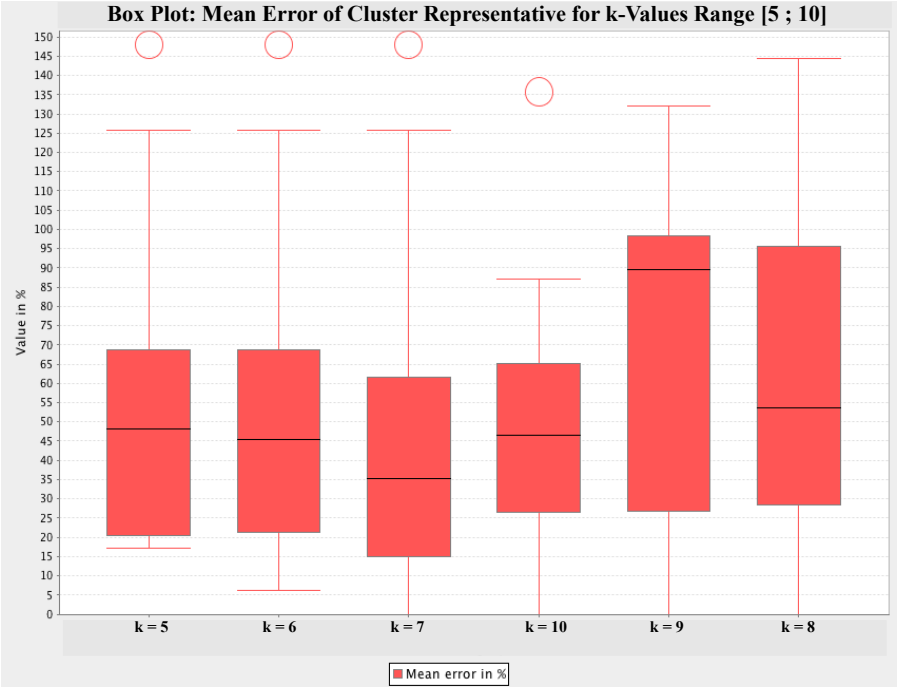

Figure 9.1: *Application Scenario 1* for data set **Basis 13-15**

9.2 Experiment 2: Box Plot Diagrams

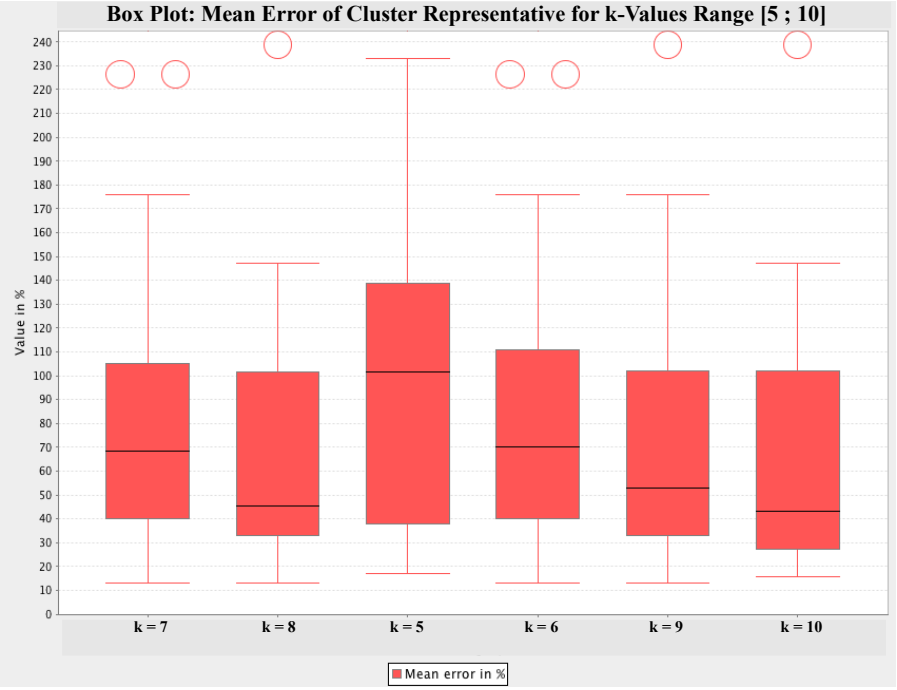

Figure 9.2: *Application Scenario 1* for data set **Basis 10-15**

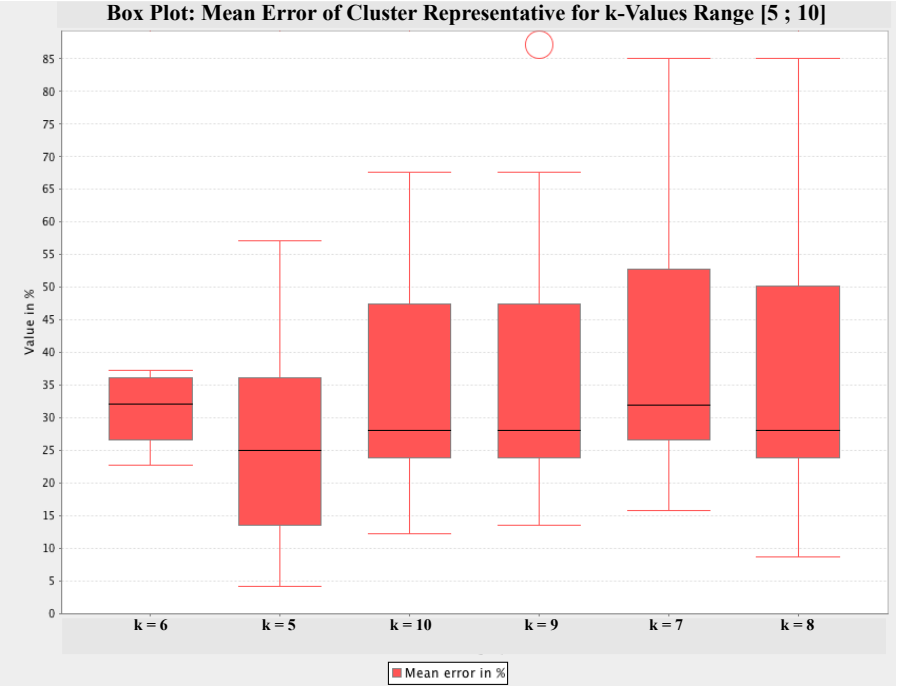

Figure 9.3: *Application Scenario 1* for data set **B 10**

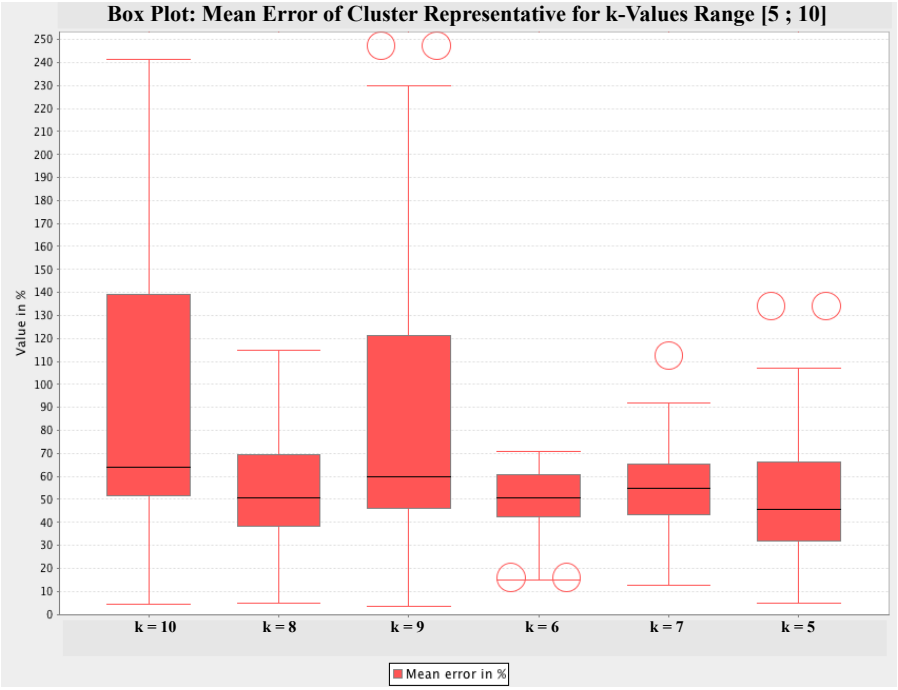

Figure 9.4: *Application Scenario 1* for data set **Basis 5-10**

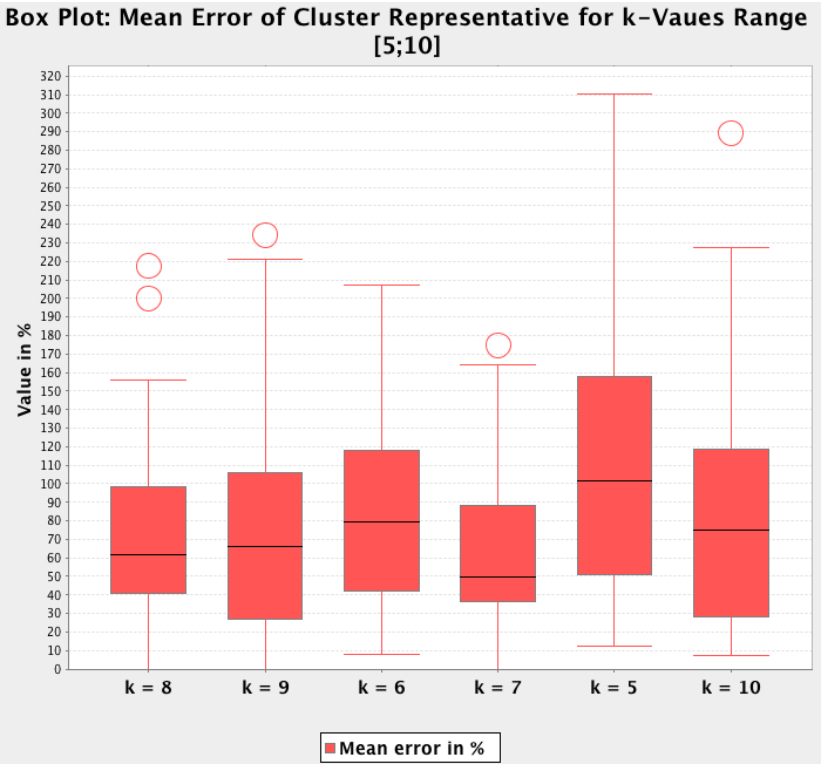

Figure 9.5: *Application Scenario 1* for data set **Basis 8-10**

Figure 9.6: *Application Scenario 1* for data set **Basis 6-8**

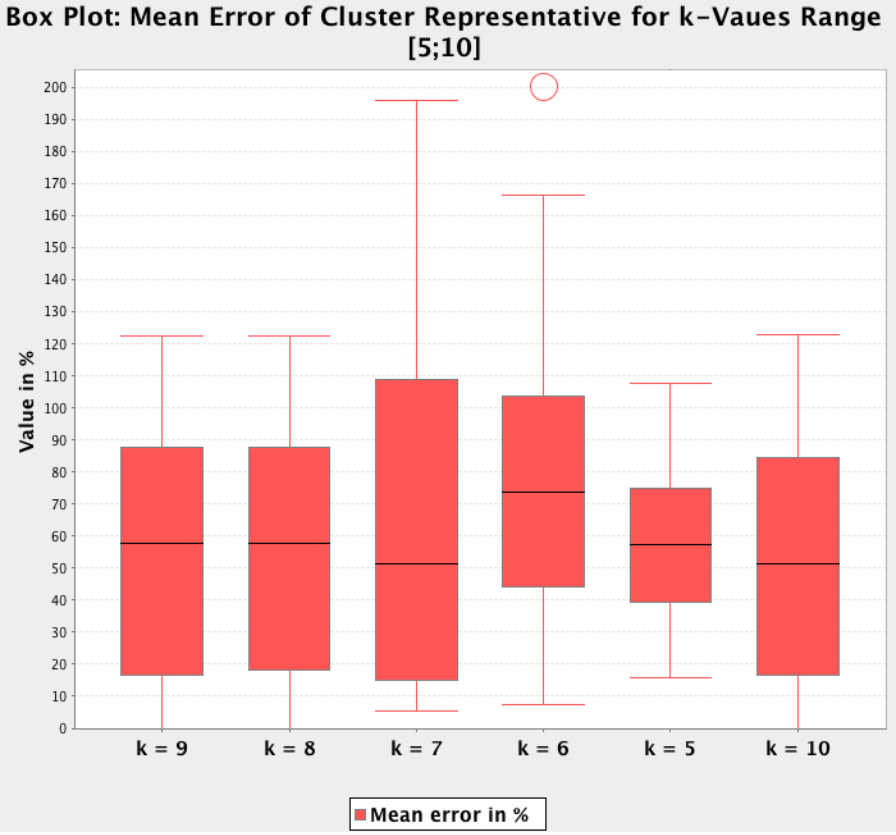

Figure 9.7: *Application Scenario 1* for data set **Basis 6**

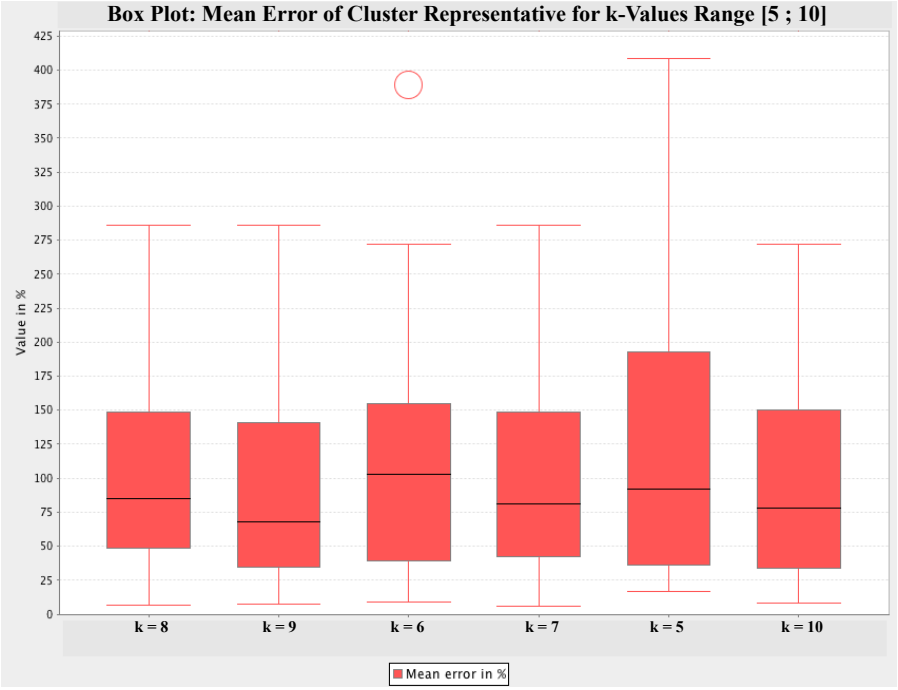

Figure 9.8: *Application Scenario 1* for data set **Basis 8-15**

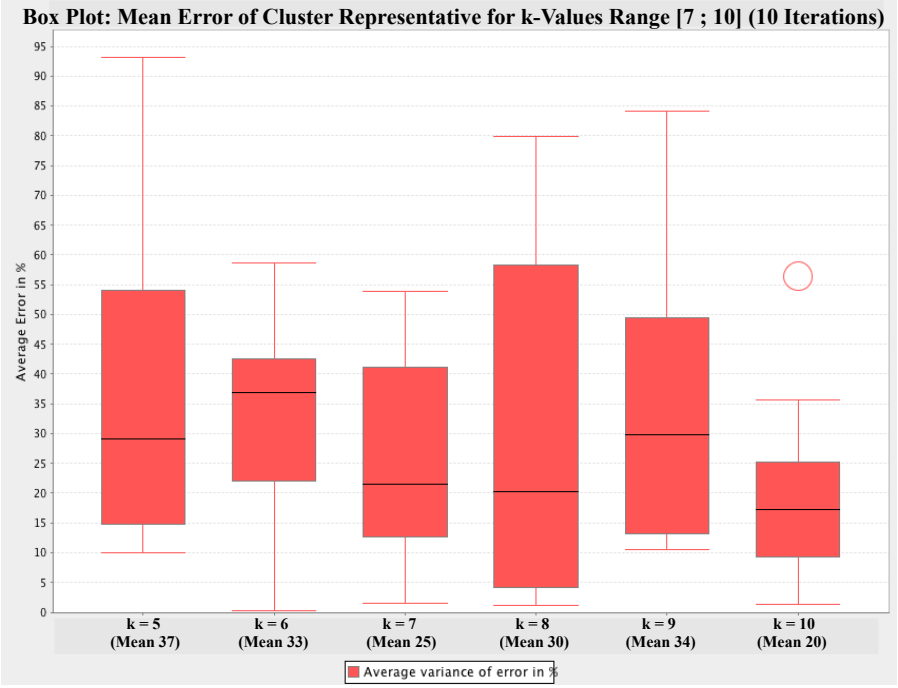

Figure 9.9: *Application Scenario 2* for data set **B 14-15**

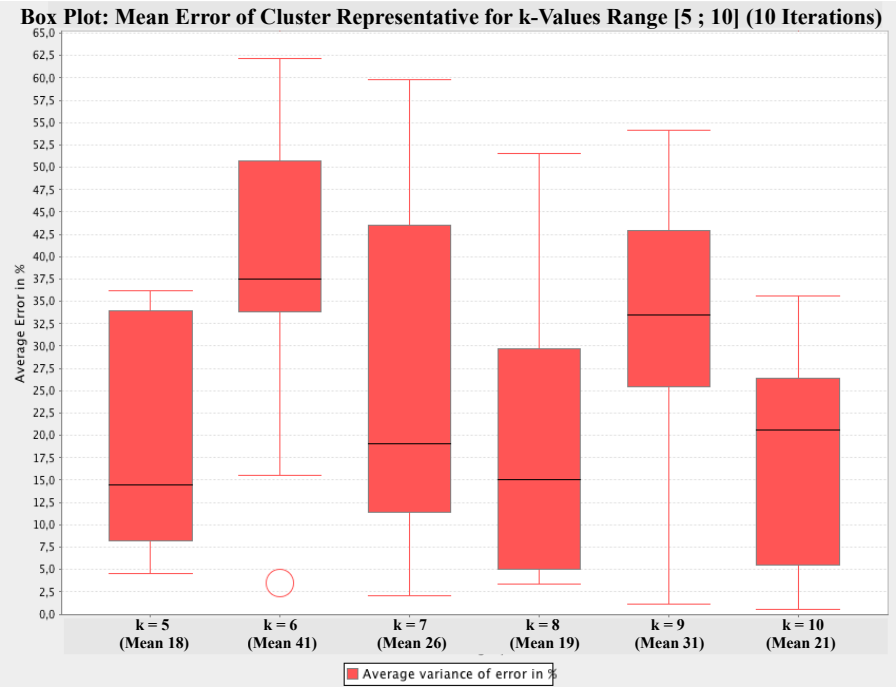

Figure 9.10: *Application Scenario 2* for data set **B 10-15**

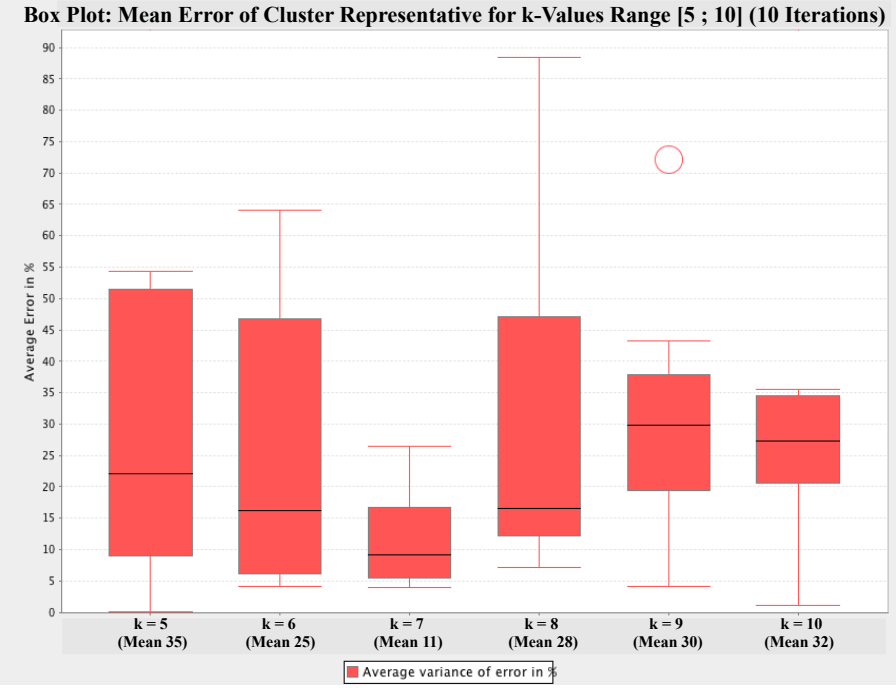

Figure 9.11: *Application Scenario 2* for data set **B 5-10**

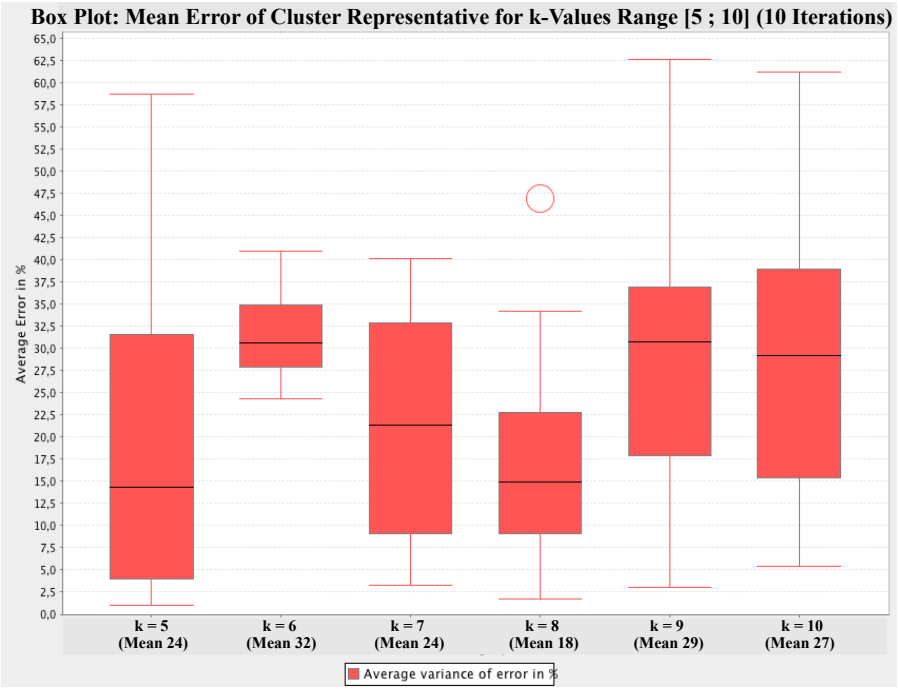

Figure 9.12: *Application Scenario 2* for data set **B 8**

Bibliography

[Abe10a] Michael Abernethy. Data mining with WEKA, Part 2: Classification and clustering. http://www.ibm.com/developerworks/opensource/library/os-weka2/index.html, 2010. [Online; accessed 15-March-2015].

[Abe10b] Michael Abernethy. Data mining with WEKA,Part 1: Introduction and regression. http://www.ibm.com/developerworks/opensource/library/os-weka1/index.html, 2010. [Online; accessed 15-March-2015].

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2003.

[Bel14] J. Bell. *Machine Learning: Hands-On for Developers and Technical Professionals*. John Wiley & Sons, Inc., 10475 Crosspoint Boulevard, Indianapolis, IN, USA, 2014.

[BFK+13] Remco R. Bouckaert, Eibe Frank, Richard Kirkby, Peter Reutemann, Alex Seewald, and David Scuse. WEKA Manual for Version 3-7-8. http://sourceforge.net/projects/weka/files/documentation/3.7.x/WekaManual-3-7-8.pdf, 2013. [Online; accessed 7-March-2015].

[BR90] Victor R. Basili and H. D. Rombach. Towards a comprehensive framework for reuse: Model-based reuse characterization schemes. Technical Report ADA222498, Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park, MD 20742, April 1990. Technical rept.

[CD13] Cagatay Catal and Banu Diri. A Fault Detection Strategy for Software Projects. *Technical Gazette*, 20(1):1–7, Feb 2013.

[CLP+13] Gerardo Canfora, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Multi-objective cross-project defect prediction. In *ICST*, pages 252–261. IEEE, 2013.

[Das02] Sanjoy Dasgupta. Performance guarantees for hierarchical clustering. In *15th Annual Conference on Computational Learning Theory*, pages 351–363. Springer, 2002.

[DT96] Merlin Dorfman and Richard H. Thayer. *Software engineering*. IEEE, 1996.

[EE08] Karim O. Elish and Mahmoud O. Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, 2008.

[Ele90] Inc Electronics Engingeers. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 121990:84, 1990.

[ES00] Martin Ester and Jörg Sander. *Knowledge Discovery in Databases: Techniken und Anwendungen*. Springer Berlin Heidelberg, 1 edition, 2000.

[Evo12] Evolus. Pencil Project. `http://pencil.evolus.vn/Default.html`, 2012. [Online; accessed 11-April-2015].

[faSQMM15] IEEE Std 1061-1998 IEEE Standard for a Software Quality Metrics Methodology. IEEE Standards Definition Database Search - S. `http://dictionary.ieee.org/index/s-14.html`, 2015. [Online; accessed 7-April-2015].

[Fis87] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[FK02] Quentin Fleming and Joel Koppelman. *Earned Value Project Management, Second Edition*. Project Management Institute, second edition, 2002.

[Fow00] Martin Fowler. Plain Old Java Object (POJO). `http://www.martinfowler.com/bliki/POJO.html`, 2000. [Online; accessed 14-April-2015].

[FPR00] Marcus Fontoura, Wolfgang Pree, and Bernhard Rumpe. *The Uml Profile for Framework Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[FS97] Mohamed Fayad and Douglas C. Schmidt. Object-oriented Application Frameworks. *Commun. ACM*, 40(10):32–38, October 1997.

[GCS03] L. Guo, Bojan Cukic, and H. Singh. Predicting fault prone modules by the Dempster-Shafer belief networks. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 249–252, Oct 2003.

[GLF90] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1990.

[Gui14] Gui Prototyping Tools. `http://c2.com/cgi/wiki?GuiPrototypingTools`, 2014. [Online; accessed 11-April-2015].

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[HGNW10] Steffen Herbold, Jens Grabowski, Helmut Neukirchen, and Stephan Waack. Retrospective Analysis of Software Projects using k-Means Clustering. *Softwaretechnik-Trends*, 30(2), 2010.

[HGW11] Steffen Herbold, Jens Grabowski, and Stephan Waack. Retrospective Project Analysis Using the Expectation-Maximization Clustering Algorithm. *Third International Conference on Advances in System Testing and Validation Lifecycle (VALID 2011)*, (Figure 1):58–63, 2011.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Academic Press, London, United Kingdom, 1 edition, 2001.

[HK06] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Elsevier Inc., San Francisco, CA, USA, 2nd edition, 2006.

[HS85] Hochbaum and Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[HSM01] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of Data Mining*. MIT Press, Cambridge, MA, London, England, 1 edition, 2001.

[Ins04] Project Management Institute. *Practice Standard for Earned Value Management*. Project Management Institute, Incorporated, 2004.

[JM10] Marian Jureczko and Lech Madeyski. Towards identifying software project clusters with regard to defect prediction. In Tim Menzies and Gunes Koru, editors, *PROMISE*, page 9. ACM, 2010.

[Ker09] Harold R. Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons, Inc., 10 edition, 2009.

[LL10] J. Ludewig and H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. Dpunkt.Verlag GmbH, Heidelberg, 3 edition, 2010.

[MNU00] A. McCallum, K. Nigam, and L.H. Ungar. Efficient clustering of high dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD internation conference on knowledge discovery and data mining ACM-SIAM symposium on Discrete algorithms*, pages 169–178, 2000.

[MSLM04] Atchara Mahaweerawat, Peraphon Sophatsathit, Chidchanok Lursinsap, and Petr Musilek. Fault Prediction in Object-Oriented Software Using Neural Network Techniques. In *Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University*, pages 27–34, 2004.

[MZC+14] Baojun Ma, Huaping Zhang, Guoqing Chen, Yanping Zhao, and Bart Baesens. Investigating associative classification for software fault prediction: An experimental perspective. *International Journal of Software Engineering and Knowledge Engineering*, 24(1):61–90, 2014.

[O'H08] Tom O'Haver. Smoothing. `http://terpconnect.umd.edu/~toh/spectrum/Smoothing.html`, 2008. [Online; accessed 12-April-2015].

[Ora13] Oracle. The Java EE 6 Tutorial: JavaServer Faces Technology. `http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf`, 2013. [Online; accessed 14-April-2015].

[Pat09] Microsoft Patterns. *Microsoft Application Architecture Guide*. Microsoft Press, 2nd edition, 2009.

[PMI04] PMI, editor. *A Guide to the Project Management Body of Knowledge (PMBOK Guide): An American National Standard ANSI/PMI*. Project Management Institute, Newtown Square, PA, 3 edition, 2004.

[Pre95a] Wolfgang Pree. *Design Patterns for Object-oriented Software Development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[Pre95b] Wolfgang Pree. Hot-spot-driven framework development. In *Summer School on Reusable Architectures in Object-Oriented software Development*, pages 123–127. ACM, 1995.

[Pre95c] Wolfgang Pree. Visual object-oriented programming. chapter Framework Development and Reuse Support, pages 253–267. Manning Publications Co., Greenwich, CT, USA, 1995.

[QT03] Tong-Seng Quah and Mie Mie Thet Thwin. Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics. In *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands*, page 116. IEEE Computer Society, 2003.

[RB10] Andrew Lee Rubinger and Bill Burke. *Enterprise JavaBeans 3.1*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 6th edition, 2010.

[RH13] Rudolf Ramler and Johannes Himmelbauer. Noise in bug report data and the impact on defect prediction results. In *IWSM/Mensura*, pages 173–180. IEEE, 2013.

[RWS+09] Rudolf Ramler, Klaus Wolfmaier, Erwin Stauder, Felix Kossak, and Thomas Natschläger. Key questions in building defect prediction models in practice. In Frank Bomarius, Markku Oivo, Päivi Jaring, and Pekka Abrahamsson, editors, *PROFES*, volume 32 of *Lecture Notes in Business Information Processing*, pages 14–27. Springer, 2009.

[SFT02] Noam Slonim, Nir Friedman, and Naftali Tishby. Unsupervised document classification using sequential information maximization. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 129–136, New York, NY, USA, 2002. ACM.

[SGM02] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[SM09] Inc. Sun Microsystems. JSR 318: Enterprise JavaBeans™,Version 3.1 EJB Core Contracts and Requirements. `http://download.oracle.com/otn-pub/jcp/ejb-3.1-fr-eval-oth-JSpec/ejb-3_1-fr-spec.pdf?AuthParam=1429001965_5b7539451c3887cb99f9131223612520`, 2009. [Online; accessed 14-April-2015].

[Sny03] Carolyn Snyder. *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann Pub, 2003.

[Som06] Ian Sommerville. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 8 edition, 2006.

[Son] Sonar. SonarQube. `http://www.sonarqube.org`. [Online; accessed 30-April-2015].

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Elsevier Inc., San Francisco, CA, USA, 2nd edition, 2005.

[WRB08] Dindin Wahyudin, Rudolf Ramler, and Stefan Biffl. A framework for defect prediction in specific software project contexts. In Zbigniew Huzar, Radek Kocí, Bertrand Meyer, Bartosz Walter, and Jaroslav Zendulka, editors, *CEE-SET*, volume 4980 of *Lecture Notes in Computer Science*, pages 261–274. Springer, 2008.

[WWM10] Xiaoxu Wang, Chaoying Wu, and Lin Ma. Software project schedule variance prediction using bayesian network. In *Advanced Management Science (ICAMS), 2010 IEEE International Conference on*, volume 2, pages 26–30, July 2010.

[You00] Edward Yourdon. *Software Engineering Project Management*. Wiley-IEEE Computer Society Press, Los Alamitos, CA, USA, 2nd edition, 2000.

[ZNG+09] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In Hans van Vliet and Valérie Issarny, editors, *ESEC/SIGSOFT FSE*, pages 91–100. ACM, 2009.