I hereby declare that this PhD thesis was written by myself.
I have quoted all the references I have drawn upon.

Ostrava, 28th June 2017
Supervisor of PHD Thesis

Prof. Ing. Ivo Vondrák, CSc.
ivo.vondrak@vsb.cz

Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba
Czech Republic

http://www.cs.vsb.cz
http://fei.vsb.cz
http://www.vsb.cz

PHD THESIS
Copyright © 2017 Radoslav Štrba
Acknowledgements
First of all, I would like to thank my thesis supervisor, Professor Ivo Vondrák for opportunity to work on this interesting research topic under his guidance and also for his help with obtaining the data from a software company for my experiments published in this thesis.

I would like to thank my colleagues from the Research Group of Software Engineering (RGSE) at Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VŠB – Technical University of Ostrava. I would like to thank to Doctor Svatopluk Štolfa and Jakub Štolfa for their previous work, suggestions, advices and cooperation in a field of effort estimation. My thanks also go to Doctor David Ježek for his explanation and description of the additional data for my experiments.

Finally, I am also very thankful to my close family, especially my mother Anna for her patience and encouragement, during this study-period of my life and to my sister Kristína for the cooperation on the part of my research related to usage of statistical methods. Last but not least, I would like to thank to my girlfriend Mirka for her support, patience and smile.
Abstrakt


Klíčová slova

softwarový proces, odhad nákladů, softwarový požadavek, Naïve Bayes, umělá neuronová síť, klasifikace, exploratorní analýza, SOM

Abstract

Effort overruns is common problem in software development. This dissertation thesis is focused on design of new advanced method for software process support in early phase of software development. In particular, this method helps to improve software development process using results of classification of software requirements. Those requirements are experimentally classified using machine-learning methods like neural network or Naïve Bayes classifier. Results of classification help to project managers or analysts make estimations of time duration of work more accurately. Part of this PhD thesis provides a guideline for software effort estimation. Companies should be able to deploy, configure and use proposed methodology using the guideline. An estimation process should be also improving continuously.

Keywords

software process, effort estimation, software requirement, Naïve Bayes, artificial neural network, classification, exploratory analysis, SOM
Contents

1. **INTRODUCTION** ........................................................................................................ 9
   1.1. **THESIS GOAL** .................................................................................................... 10
   1.2. **ORGANIZATION OF THESIS** ........................................................................ 10

2. **STATE OF THE ART** ............................................................................................... 12
   2.1. **SOFTWARE PROCESS SUPPORT** ................................................................. 12
   2.2. **INTRODUCTION TO SOFTWARE EFFORT ESTIMATION** ............................. 14
   2.3. **APPROACHES AND MODELS FOR EFFORT ESTIMATION** ....................... 15
       1.2.1 **Algorithmic Models** ................................................................................ 15
       1.2.2 **Expert Judgment and Estimation by Analogy** ........................................ 16
       1.2.3 **Soft Computing Models** ......................................................................... 16
   2.4. **COMPARISON OF MODELS AND APPROACHES** ........................................ 16
   2.5. **EFFORT ESTIMATION SUPPORTED BY MACHINE-LEARNING** ................. 17

3. **CLASSIFICATION AS A SUPPORTIVE TECHNIQUE** .............................................. 18
   3.1. **LOGICAL METHODS** ..................................................................................... 18
   3.2. **STATISTICAL METHODS** .............................................................................. 18
   3.3. **ARTIFICIAL INTELLIGENCE** ......................................................................... 21
   3.4. **EXPERIMENT – CLASSIFICATION OF USE CASES** ..................................... 26
       4.5.1 **Parameterization of Use Cases** ................................................................ 26
       4.5.2 **Classification using Neural Network** ..................................................... 29
       4.5.3 **Classification using Naïve Bayes** ............................................................ 34
       4.5.4 **Summary of Results** .............................................................................. 37

4. **EXPLORATORY ANALYSIS OF SOFTWARE REQUIREMENTS** ......................... 39
   4.1. **PARAMETERIZATION OF SOFTWARE REQUIREMENTS** ............................. 39
   4.2. **EXPLORATORY ANALYSIS** ............................................................................ 41
   4.3. **VISUALIZATION OF RESULTS OF THE CLUSTERING** ................................. 47
   4.4. **RESULTS SUMMARY OF EXPLORATORY DATA ANALYSIS** ....................... 48

5. **PROPOSED METHOD FOR EFFORT ESTIMATION** .............................................. 50
   5.1. **DATA MODEL, INPUTS AND OUTPUTS** ........................................................ 54
   5.2. **DEPLOYMENT, CONFIGURATION AND USAGE OF METHODOLOGY** .......... 57
   5.3. **GUIDELINE AND BEST PRACTICES** ............................................................ 63
   5.4. **EXPERIMENT - EVALUATION OF PROPOSED METHOD** ............................. 65
       4.5.5 **Classification of Software Requirements** ............................................... 68
       4.2.1 **Results of Classification** ........................................................................ 70

6. **SUMMARY AND CONCLUSION** .......................................................................... 73
List of Abbreviations and Symbols

AI – Artificial Intelligence
CRM – Customer Relationship Management
ERP – Enterprise Resource Planning
FPA – Function Point Analysis
MSE – Mean Squared Error
NB – Naïve Bayes
NN – Neural Network
PCA – Principal Component Analysis
RUP – Rational Unified Process
SOM – Self-Organizing Map
SPI – Software Process Improvement
SQL – Structured Query Language
SVM – Support Vector Machine
UCP – Use Case Point
List of Tables

Table 1: Example of parameterized use cases. ............................................................. 28
Table 2: Results of online training. ........................................................................ 32
Table 3: Results of batch training. ........................................................................... 32
Table 4: Results of online training using testing set from next year. .................... 33
Table 5: Results of batch training using testing set from text year. ......................... 33
Table 6: Results of training using testing set of 10 use cases. ............................... 36
Table 7: Results of training using testing set of 10 use cases. ............................... 36
Table 8: Results of testing using set of 10 testing use cases from the current dataset .. 37
Table 9: Results of testing using set of all use cases from the next dataset. ............. 38
Table 10: Example of requirement entity, including parameters with example values .... 40
Table 11: Table of values with same unit (hours) from Figure 9. ............................ 42
Table 12: Table of normalized values from Figure 10. ........................................... 43
Table 13: Artefact – Template for software requirement. ....................................... 53
Table 14: Example of transformation – type of requirement. .................................... 62
List of Figures

Figure 1 Software Development Methodologies [9]……………………………………………… 13
Figure 2: Activity diagram of requirements engineering process .......................................... 14
Figure 3: Example of feed-forward neural network topology ........................................... 23
Figure 4: Tangent Hyperbolic function. ............................................................................. 24
Figure 5: Example of the parameterization and transformation of use case scenario. ........ 27
Figure 6: Overview of the classification process using neural network ............................ 30
Figure 7: State-chart diagram describes states of Use Case after classification process ...... 31
Figure 8: Overview of the classification process using Naive Bayes Classifier ............... 34
Figure 9: Box-plot and Whisker-plot of variables with same units (hours) - R studio .. 41
Figure 10: Box-plot and Whisker-plot of normalized variables - output R studio .......... 42
Figure 11: Correlation circle of PCA ................................................................................. 43
Figure 12: Correlation matrix ............................................................................................. 44
Figure 13: Hexagonal topology of SOM (25 × 25) constructed by MATLAB environment.. 45
Figure 14: Component plane of Kohonen’s layer of SOM ................................................. 47
Figure 15: Activity diagram – Overview of method form the top-level perspective ......... 51
Figure 16: Class diagram - Data model of example project management application ....... 54
Figure 17: Daily-usage of Methodology ............................................................................ 57
Figure 18: Lifecycle of Methodology ................................................................................ 58
Figure 19: Activity diagram – Steps of proposed methodology ........................................ 60
Figure 20: Example of Effort Estimation .......................................................................... 64
Figure 21: Class diagram – Data model ............................................................................. 66
Figure 22: Training State Plot – Gradient ......................................................................... 69
Figure 23: Training State Plot – Validation Checks ............................................................ 69
Figure 24: Validation Performance .................................................................................... 70
Figure 25: Confusion Matrixes ......................................................................................... 71
1. Introduction

“How to improve the accuracy of software development process effort estimations?” – this is important question for project managers in software companies. Accuracy of effort estimation depends on many external and internal factors. For example, an amount of work, risk factors, testing level, remote work and other important parameters need to be considered. It’s not possible to estimate time of work on development tasks with 100% accuracy. However, the number of underestimated tasks can be significantly reduced. [1, 2]

Effort Estimation of software projects has become an important task in software engineering and project management. Old estimation methods that have been used for prediction of project costs that have been developed using procedural languages are becoming inappropriate methods for estimation of the more recent projects, which are created with usage of some object-oriented languages. It calls for more advanced and sophisticated approaches and for new supportive methods for effort estimation of software projects.

“In 2013, The Standish Group states that 43% of software development projects were delivered late or over the budget in The Chaos manifesto 2013“ [3]. Those results show another increase in project success rates, with 39% of all projects succeeding. Those projects have been delivered on time, on budget, as well as with required features and functions. Finally 18% of projects failed because they have been cancelled prior to completion or delivered and never used. Some of reasons of project failure are for example, lack of estimation of the staff’s skill level, misunderstanding the requirements or improper software size estimation. “Another study presented by The International Society of Parametric Analysis to determine the main factors that lead to project failures. Those factors include uncertainty of software system and software requirements, unskilled estimators, limitation of budget for project, optimism in software estimations, ignoring historical data or unrealistic estimations.” [2] In a few words, some software projects fail because of the inaccuracy of software estimations and misunderstanding or incompleteness of the software requirements. This fact motivated researchers to focus on research related to improvement of software development effort estimations for better software size and effort assessment.

The main idea of this dissertation thesis is to provide method for support of software process. Particularly, the supported activity of software process is effort estimation. It’s supported by machine-learning methods that are not so common in this area. Those methods and techniques are bringing a new point of view on effort estimation. Proposed method can help to project managers or analysts to estimate complexity of projects and the risk of additional work for existing projects based on the analysis of requirements. The proposed method uses knowledge base with historical data. It also provides support of decisions in the form of a probability given by estimation of work-time (time of work) in the project. In nutshell, also the guideline for usage of proposed method for support of software effort
estimations based on classification of software requirements is provided. For classification task, a feed-forward Neural Network architecture with Back-propagation training algorithm is applied within the scope of the proposed method. [4]

1.1. Thesis Goal

The main goal of this dissertation thesis is to show usage of artificial intelligence or machine learning for support of software effort estimation. It’s important phase of software development process. An appropriate machine-learning technique should be selected and used for support of effort estimation.

Particular Goals:

- The first particular goal of thesis is to provide state of the art based on overview and description of existing methods and approaches used for field of software effort estimation.
- Second particular goal is to select appropriate machine-learning classifier and find optimal configuration of selected classifier. Then, a method for the support of effort estimation that is using selected classifier should be proposed.
- Finally, the last goal of thesis is to perform experiments using machine-learning classifiers for classification of software requirements and evaluate results.

1.2. Organization of thesis

This dissertation thesis provides insight into the field of software effort estimation. Advanced method for software process support especially for effort estimation is proposed. The thesis is divided into three main parts, which are divided into six sections. The first part provides information about effort estimation in early stage of software development. This part also provides an overview of widely used approaches and comparison of those approaches. Pros and cons particular approaches and models are described as well. Second part of thesis consists of details about parameterization of software requirements and data pre-processing. Selected technique is evaluated by classification of software requirements into categories according to estimated time of work. The first classification experiment is done using by Neural Network as artificial intelligence method and Naive Bayes (NB) as statistical method. The third part introduces proposed methodology and shows results of evaluation. That evaluation described by second classification experiment. In nutshell, this part provides description of proposed methodology with guideline for deployment and usage. Examples,
best practices and results of evaluation of the method are also provided. Previously mentioned parts of this thesis are further divided into six following sections.

The first Section 1 is Introduction section that provides general information about thesis purpose. Information about effort estimation and also overview of reasons behind project failures is provided.

Section 2 is called as State of the Art, provides introduction into software process support, software process improvement and particularly into the field of effort estimation. This section provides complex information about approaches and models used for software effort estimation. It also shows brief description of these approaches its comparison based on pros and cons. Section 2 also describes usage of machine-learning techniques for support of older effort estimation approaches.

Section 3 introduces classification as a supportive technique for effort estimation. The section briefly describes three different approaches for classification. Logical, statistical and artificial intelligence methods are described. Experiment shows that appropriate classifier for classification of software requirements in form of use cases is feed-forward neural network with backpropagation training algorithm. Furthermore, these results of other classifiers are compared also with results of the classification based the feed-forward neural network architecture.

Section 4 shows results of exploratory analysis of software requirements and other data from real software company. Statistical method – Principal Component Analysis (PCA), and neural network – Kohonen’s Self-Organizing Map (SOM) have been used. Both methods provide different point of view on the data.

Fifth Section 5 proposes method for software effort estimation based on classification of historical data obtained from a database of project management tool. This approach for estimations combines expert knowledge with artificial intelligence. Proposed method is detailed described and evaluated by experiment. The experiment uses real data from project the database of management tool of the software company.

Finally, the conclusion Section 6 includes summarization of reached goals and results of proposal of this thesis. The section also closes the dissertation thesis with paragraphs named “pros and cons” of the proposed method and “future plans” in field of software effort estimation and usage of proposed methodology.
2. State of the Art

2.1. Software Process Support

*Software process* (In other words "software development methodology") is sequence of steps or activities from initial inception of customer to the release of the created product. [5]

Software process includes analysis, design, programming, testing, configuration management, and other sub-processes that are used to reach a goal, which is the creation of software product. In other words, software process is the sequence of connected activities, executed to develop a software product in expected time, quality and budget. There are few disciplines related to software process. One of them is the discipline called - process engineering. This discipline includes other disciplines for support of software process like software process improvement, modelling and planning, or measurement of the process. [6]

The software process improvement (SPI) is discipline that plays an important role in business environment. Software companies have invested large sums of money to the quality improvement of their software processes. The goal of SPI is to allow develop and maintain a software system by the most efficient way. Particular activities of software process, like elicitation of requirements, analysis and design of software system, implementation, etc., which are essential activities of the whole software process consist of many tasks. Those activities include also some creative, administrative, or communication tasks as supportive activities. Each of those tasks requires a specific and narrow knowledge. The important administrative task for successfully finished software product is planning of resources and estimation of the effort. Specific knowledge of expert in certain domain is necessary as well. A historical data, implementation environment, type of software, skills and number of developers needs to be also taken into the account. The historical data from the software development process is a very important artefact for future estimations of working-time, future planning and also for development of a good software product. Artefacts that are used during software development should be created according to pre-described rules, defined steps and templates. Description of rules and steps of artefacts is inseparable part of software process support.

Next part of software process support is description of methods that are used in a software company. Each software company uses individual approaches for specification of requirements, effort estimation and implementation of software product. Each developer should follow up early-defined best practices and lessons learned that are typically presented in the form of case studies or methods. Some of best practices are the same for years, but some of them are constantly changing. Combination of expert knowledge, recommended artefacts and best practices is crucial for the success of a software company.
In addition, innovative approach for the support of human decisions is usage of machine-learning techniques. In field of software effort estimation, the effort required to develop a new software project is estimated by taking important parameters and fill them into estimation tool. Project manager or analyst usually performs this activity. The goal of this activity is to make estimation as accurate as possible. Software effort estimation is one of key concerns of software process support. [6] [7] [8]

Figure 1 bellow shows three widely-used software process models. The first model is called Waterfall. This model includes activities like requirements specification, analysis and design, implementation and testing. Those activities have been improved and used also in Rational Unified Process (RUP), which defines best practices for software development. These best practices have been inherited by agile methodologies. The mutual goal of all software development methodologies is to provide high quality software product within budget and in time. This calls for usage of advanced methods for software process support. The method for support of software effort estimation is a part of “Requirements” activity. [5] We can go deeper into the effort estimation issue in following chapter.

Figure 1 Software Development Methodologies [9]
2.2. Introduction to Software Effort Estimation

“Requirements” - engineering process consists of several activities. If requirements will be specified for a new system, then it is important to analyse a problem. It means, that an agreement on a statement of the addressed problem should be captured. Stakeholders, boundaries and constraints of the system should be identified. If requirements will be specified for existing software system, then it is important the understanding of stakeholder’s needs. Stakeholder’s requests and clear understanding of needs of user should be gathered.

Process continues by definition of software system. The system features that are required by stakeholders should be established. Now, actors and use cases of the system are identified for each of key features. Manage the scope of the system is an appropriate activity for software effort estimation. The functional and non-functional requirements should be collected and written use cases should be prioritized according to customer needs. The system, which is developed by following up those steps is ready to be delivered on expected time and within the budget.

![Activity diagram of requirements engineering process](image)

Software effort estimation can be done at any stage within the process of requirements engineering. However, performing estimation in the early stage of software development, such as requirements elicitation means that requirements for the software system are not complete and more assumptions will need to be made in the estimation process. This could lead to poor results. [10] There is needed to find right stage within Requirements Engineering Process, in which effort estimation can be done.
2.3. Approaches and Models for Effort Estimation

“In the beginning of the 1980s, Jenkins, Naumann and Wetherbe [11] conducted a large empirical investigation. The study focused on the early stages of system development.” [2] “Next, in early 90s, Heemstra presented the basic ideas why, when and how to estimate projects in paper” [4] “Software cost estimation. In Information and Software Technology” [12]. This section speaks about importance of estimation of the projects. Proper software effort estimation is activity which is required in every software development life cycle. Several features offered by object-oriented programming concept such as Encapsulation, Inheritance, Polymorphism, Abstraction and Coupling play an important role to manage the development process [13]. Currently used models for software development effort estimation can be divided into three categories. The first category is called algorithmic models. Second category is expert judgment and estimation by analogy. Third category is soft computing models. All mentioned categories are better described in next paragraphs. [10], [14]

1.2.1 Algorithmic Models

Algorithmic models such as, COCOMO, Function Point Analysis and Use Case Point have been proven unsatisfactory for estimating cost and effort because the lines of code and function point are both used for procedural oriented paradigm [15]. COCOMO and Function Point Analysis have certain limitations. The lines of code are dependent on the type of programming language and the Function Point Analysis depends on human decisions.

“The COCOMO methodology computes effort as a function of program size and set of cost drivers on separate project phases.” [4] The name of the model, which was originally developed by Dr. Barry Boehm and published in 1981 [16], is Constructive Cost Model was known as COCOMO 81. COCOMO uses a simple regression formula.

“Function Point Analysis method not consistently provides accurate project cost and effort estimates” [17] [18]. Allan Albrecht proposed the method in 1979 [19]. Function Points measure the functionality of software as opposed to source lines of codes, which measures the physical components of software. [20] There are a few methods to count function points but the standard method is the one that is maintained by the Function Points Analysis, which is based on the International Function Point Users Group [21].

The Use Case Point (UCP) model was proposed by Gustav Karner in 1993 model relies on the use case diagram to estimate the effort of a given software product. [22] It helps in providing more accurate effort estimation from design phase of software development life cycle. “UCP is measured by counting the number of use cases and the number of actors, each multiplied by its complexity factors. Use cases and actors are classified into three categories (complexity values). These include simple, average and complex.” [23] One of the
limitations of UCP is that the software effort equation is not well accepted by software estimators because it assumes that the relation. [10],[24, 25], [14]

1.2.2 Expert Judgment and Estimation by Analogy

Some of methods are pretty depended on knowledge of people. One of those methods is expert judgment, which involves consultations with a group of experts in certain domain to use their experiences for proposal of estimations of the project. [10] [26] Expert judgement is very similar to estimation by analogy. It’s a method, which uses a comparison of the proposed project with similar projects developed in the past. Estimation by analogy is little bit different systematic type of expert judgment approach. Since experts look for analogies in this case. The main advantage of this method is that estimators use to use their knowledge for estimation of new projects based on actually finished projects. The main disadvantage of estimation by analogy is that companies are required to maintain a well-designed repository of knowledge and information about duration and details of finished projects. Moreover, companies should have a good number of finished projects from the past, if they want to use this approach. In short, this method can’t be deployed and used inside an environment of new companies. [10, 27, 28]

1.2.3 Soft Computing Models

Group of soft computing models includes Neural Network, Fuzzy Logic, and Genetic Algorithms etc. Other models are for example Self Organizing Maps (SOM), Support Vector Machine (SVM) or Fuzzy Rules. [10] Appropriate models are also hybrid models like neural and fuzzy models. Soft computing models can be applied in two main situations:

- First situation is when these models can be applied as standalone models that take several inputs such as software size and productivity, then provide an output such as software effort.
- Second situation is when these models can be used for calibration of some parameters or weights of algorithmic models such as COCOMO parameters and function point model weights. “Soft computing models can also be used with estimation by analogy to increase the accuracy of estimation.” [10]

More detailed information about experiments using SOM, SVM, Fuzzy Rules or Neural Networks are better described in papers [29, 30], [31].

2.4. Comparison of Models and Approaches

Results of study “A Comparison of Size Estimation Techniques Applied Early in the Life Cycle” using function point analysis (FPA) shows that the average deviation between the
estimated and the actual value is about 10%. [32] Anthony Pengelly shows in his research that accuracy of COCOMO can has very similar accuracy of estimations like FPA, but it also depends on many configuration- parameters of that model. [33], [2]

Authors of other study which is named “Evaluating different families of prediction methods for estimating software project outcomes” [34] discuss about usage of artificial intelligence and machine-learning methods for classification in field of effort estimation. This field of research is close to methodology, which is proposed in this dissertation thesis. Average accuracy of experiments performed in mentioned study is about 90%. Average difference for very similar method like is described in that study is about 10%. Accuracy can be higher than 90% for some specific cases or in more ideal situations.

The comparison shows that the difference between estimated and actual values depends on complexity and quality of available information from software development. That difference also depends on breadth of knowledge database and amount and quality of historical data. The best results can be obtained by combining standard approaches for software effort estimation with soft computing techniques (including artificial intelligence and machine-learning algorithms using historical data). Then the accuracy of decisions and prediction can fluctuate between 90% and 93% in best cases. [34]

2.5. Effort Estimation Supported by Machine-Learning

Several techniques for support of software effort estimation are available. For example classification of software requirements by using neural networks or statistical methods. In 2012, the comparative study of supportive techniques for software effort estimation was published by IEEE – TRANSACTIONS ON SOFTWARE ENGINEERING. The title of that article is „Data Mining Techniques for Software Effort Estimation: A Comparative Study“. This study provides literature overview from year 1995 to year 2009 and the study also says that it’s very difficult to compare results of experiments, due to deferent data structure and pre-processing methods. [6].

The sentence „The conclusion of this research is that artificial intelligence models are capable of providing adequate estimation models.“ [35] is written in the article from year 1997, that compares techniques for effort estimation. Almost 20 years have passed and usage of neural networks for support of effort estimation is still very actual and interesting. Neural networks can be also used as a part of hybrid models. Example of hybrid model is combination of usage old methodology for estimations called COCOMO II with Artificial Neural Networks. [36]
3. Classification as a Supportive Technique

Techniques for classification of requirements can be also used as supportive techniques for effort estimation. Machine-learning methods help us to classify software requirements or use cases, which is useful for prediction of risk factor of improper or inaccurate estimation of working time. Classification is a process that is closely related to the pattern recognition. A neural network, which is trained for classification is designed to take input samples and classify them into groups (classes). [37, 38] In that type of requirements classification task we are interested in this thesis. It’s given a set of attributes, or features, of an object, and we want to decide to which of a number of classes it belongs. The given attributes can be filled into an input vector x. The system should be trained to classifying of software requirements. Given a set of sample patterns, where each consisting of a vector of attribute values and the corresponding class label. [39] A lot of machine-learning methods designed to perform classification in various domains is available. The methods differ much in their background. Some methods are developed in the context of mathematical logics, others in statistics or neural networks. [39], [14]

3.1. Logical Methods

This group includes popular methods of artificial intelligence, which represents knowledge as relations between logical attributes. Binary input parameters are treated directly, while numerical parameters are coded with appropriate predicates. [26] There are also ways to learn logical representations from examples, e.g. via Rule Induction. One approach is to represent class descriptions as Logical Conjunctions. Another logical representation is a Decision Tree. [39] [40] Logical classification rules may be appropriate in deterministic domains, where each input pattern can belong to only one class. If several classes have the same feature vector, the best one can do is to calculate the probability of the different classes, and select the most probable one. [39] In most methods based on logical expressions, each predicate normally depends on just one input attribute. It is possible to use more complex predicates. In addition, these predicates depend on more than only one attribute. The problem is that this approach is more complicated for representation and also more complicated to learn these predicates. [39]

3.2. Statistical Methods

The main point of the statistical methods is to use the training data to estimate the probability distribution over the whole domain, and then use this distribution to calculate probabilities of the classes given a specific input pattern. [39] [40] Three methods based on statistical distribution are presented in next paragraphs.
• **Non-parametric:** An example of a non-parametric method is the Parzen Estimator [41]. The main idea is to have one kernel density function for each training item, and add this items together. Typically the kernel function may be a multivariate Gaussian function, with the centre at the sample point.” [39]

• **Semi-parametric:** Semi-parametric models can be called as compromise between non-parametric and parametric models. It can be said that it’s the mixture of both of them. An important example here is perhaps a Mixture Model, where the regarded distribution consists of a finite (weighted) sum of some parametric distributions. [39] [42]

• **Parametric:** In this model all the parameters are in finite-dimensional parameter spaces. [39]

Naïve Bayes Classifier

The naïve Bayes (NB) classifier is not only a single algorithm for training such classifiers. NB is a family of classification algorithms based on a common principle. This principle is called Bayes rule (Thomas Bayes). It is common rule for classification problems in data mining and machine learning areas, because of the simplicity and impressive classification accuracy of that rule. Classifier is a probabilistic model that assigns class labels to problem instances, represented as vectors $X = (x_1, ..., x_n)$ of $n$ feature values where the class labels are drawn from some finite set. Given a set of variables $x$ of vector $X$, we want to construct the posterior probability for the $C$ among a finite set of possible classes $C = (c_1, ..., c_k)$ applying Bayes rule (3.1). [39, 43, 44]

$$P (c_k | x_1, ..., x_n) \propto P (x_1, ..., x_n) \times P (c_k)$$ (3.1)

$P (c_k | x_1, ..., x_n)$ is the posterior probability of specific class membership. If features are depended values, it assigns to instance probabilities $P (c_k | x_1, ..., x_n)$ for each of $k$ possible classes. The problem is when the number of features $n$ is too large or if a feature can take on a large number of possible values. [39] Applying Bayes theorem, the conditional probability can be decomposed like:

$$P (c_k | X) = \frac{P (c_k) \times P (X | c_k)}{P (X)}$$ (3.2)

In other words, the Bayes theorem from formula (3.2) can be rewritten like formula (3.3), where posterior is $P (c_k | X)$, prior is $P (c_k)$, likehood is $P (X | c_k)$ and evidence is $P (X)$.

$$posterior = \frac{prior \times likelihood}{evidence}$$ (3.3)
Given the intractable sample complexity for learning Bayesian classifiers, we must look for ways to reduce this complexity. The Naïve Bayes classifier does this by making a conditional independence assumption that dramatically reduces the number of parameters to be estimated when modelling $P(X | c_k)$, from our original $2(2n^2 - 1)$ to just $2n$.

**Definition of Conditional Independence:** Given random variables $X$, $Y$ and $Z$, we say $X$ is conditionally independent of $Y$ given $Z$, if and only if the probability distribution governing $X$ is independent of the value of $Y$ given $Z$, that is in equation (3.4). [45]

$$
(\forall i, j, k) \ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k) \quad (3.4)
$$

As is written above, the Naïve Bayes Classifier is based on Bayes rule that assumes the attributes of vector $X = (x_1, ..., x_n)$ are all conditionally independent of one another, given class $c_k$. The value of this assumption is that it dramatically simplifies the representation of $P(X | c_k)$, and the problem of estimating it from the training data. In case where $X = (x_1, x_2)$ it can be written using chain rule, where $n$ is the number of features and $k$ is number of classes. [45]

$$
P(X | c_k) \quad = \quad P(x_1, x_2 | c_k) \\
= \quad P(x_1 | x_2, c_k) P(x_2 | c_k) \\
= \quad P(x_1 | c_k) P(x_2 | c_k) \quad (3.5)
$$

The second line in equation (3.5) follows from a general property of probabilities, and the third line follows directly from definition of conditional independence (3.4). When $X$ contains $n$ attributes, which are conditionally independent of one another given $C$. With the help these information it is possible to write Bayes classification rule. [45]

$$
P(X | c_k) = \prod_{i=1}^n P(x_i | c_k) \quad (3.6)
$$

Applying Bayes classification rule above (3.6), we label a new case $X$ with a class level $C_j$ that achieves the highest posterior probability. Please note that the contribution from each feature $x_i$ can be written in several ways: [45]

$$
\frac{P(X | c_k)}{P(X | c_j)} = \frac{P(X | c_k)}{P(c_k)} \frac{P(c_k)}{P(X | c_j)} = \frac{P(c_k | X_i)}{P(c_k)} \quad (3.7)
$$
The goal is to learn a classifier that will output the probability distribution over possible values $c_k$ of class $C$, for each new instance $X$ that we ask it to classify into specific class $c_k$. [45]

$$P (c_k | X) = \frac{P (c_k) \times P (X | c_k)}{\sum_j P (c_j) \times P (X | c_j)}$$

(3.8)

**Classifier construction:** The Naïve Bayes Classifier combines probabilistic model with a decision rule. The classical way of estimating parameters in probability distributions is by maximum likelihood. This means that the estimated value of the parameter is the one that would have made the probability of the data as large as possible. If we are interested only in the most probable value of given class $C$, then we have the Naïve Bayes classification rule, where denominator does not depend on $c_k$: [45]

$$\hat{C} = \arg\max_{k \in \{1, \ldots, K\}} P(c_k) \prod_{i=1}^{n} P(x_i | c_k)$$

(3.9)

### 3.3. Artificial Intelligence

The main idea behind Artificial Intelligence Methods is to make right decisions in uncertain environment. This group of methods includes Artificial Neural Networks, simply called Neural Networks, which have ability to classify items into categories. Classification is a process that is closely related to the pattern recognition. A neural network trained for classification is designed to take input samples and classify them into groups. [37, 43] The pattern for classification is typically fed into the network as activation of a set of input units. This activation is then spread through the network via the connections, finally resulting in activation of the output units, which is then interpreted as the classification result.

Generally, showing the patterns of the training set to the network performs the training of neural network. Two kinds of the training are known. The first is supervised and second unsupervised training. Supervised training methods, where the correct class label has to be given when updating the weights. Another kind is called unsupervised (i.e. Reinforcement Learning or Clustering) in which only a global signal indicating if the answer was wrong or right is given. Nowadays, many neural network architectures are available. The most popular architecture of neural network is Multi-Layer Perceptron (also known as feed-forward architecture). Others are for example Single-Layer Perceptron, Self-Organizing Maps, Learning Vector Quantization and Recurrent Neural Networks (i.e. Hopfield Network or Boltzmann Machine). [39]
Feed-Forward Backpropagation Neural Network

The Feed-Forward is common architecture for classification purpose. This type of the architecture is usually related to supervised training algorithm called Back-Propagation. The supervised training is deduction from training data sample. [37] “Training of the neural network is the process of finding a set of weight and bias values. For a given set of inputs, the outputs produced by the neural network are very close to some target values.” [46] Historical data transformed into vectors are necessary for training of the neural network. “Back-Propagation training of the neural network searches for a set of weights and biases that most accurately predicts value from input samples. Once we have these weight and bias values, we could apply them to an upcoming dataset of items to classify them.” [46], [47, 48]

The two most useful training procedures are batch and online. The batch (offline) training means that the input data are entered into the network sequentially. After all input vectors are processed the total MSE (3.12) value is computed by comparing outputs of the neural network with the real target values. The network parameters (weights, biases) are than updated according to this error value obtained. During the online training weights are calculated and weight is changed for each individual training example. Therefore, it is possible that the error will decrease as the training samples increase. [49] [40] For more information about classification generally using feed-forward neural network and backpropagation training algorithm we refer reader to [50–52]

Most of development of neural networks today is based upon manual design and configuration. A person who has knowledge about the specific application area specifies a network architecture, configuration and activation dynamics. This state of affairs is perhaps not surprising, given that the general space of possible neural networks is so large and complex that automatically searching for an optimal network architecture may in general be computationally intractable or at least impractical for complex applications [53, 54]

Feed-forward network begins with input layer. The input layer is connected to a hidden layer, which is connected directly to the output layer. There is one hidden layer. The output layer of the neural network is what actually presents a pattern to the external environment. The number of input and output neurons is directly related to the intended use of neural network. In this case the neural network is used to solve a classification problem. Items are classified into two separate groups. We have output neuron for each group, so at all we have two output neurons. [37] More information about selected neural network architecture and activation functions has been published in the paper named “Finding an Optimal Configuration of the Feed-forward Neural Network”. [31]

Following Figure 3 provides toy example of the three-layer neural network architecture with interconnected neurons. Number of input neurons is eight and number of output neurons is two in that example.
The choice of activation functions may strongly influence the complexity and performance of neural networks. While linear functions are particularly used in input and output layers, non-linear activation functions can be used for hidden and output layers. The most common sigmoid function produces the output signal over the 0 to -1 closed range. The values 0 and 1 are obtained for only minus and plus infinities, respectively. As the output values come to close these limits, the derivations of this function decreases. The second most widely used activation function is the tangent hyperbolic function [55].

Hyperbolic tangent activation function in Figure 4, returns both positive and negative values. When graphed, the hyperbolic tangent function looks very similar to the log-sigmoid function. The important difference is that tangent hyperbolic function returns a value between -1 and +1 instead of between 0 and 1 [56].

Figure 3: Example of feed-forward neural network topology

Activation functions of hidden layer and output layer
The hyperbolic tangent activation function has been selected for hidden layer and it returns both positive and negative values. When graphed, the hyperbolic tangent function looks very similar to the log-sigmoid function. The important difference is that \( \tanh \) function returns a value between -1 and +1 instead of between 0 and 1. The algebraic expression of hyperbolic tangent activation function is below (3.10).

\[
\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = e^{2x} - 1 \quad \frac{1}{1 + e^{-2x}}
\]

(3.10)

The most usual kind of classification problem is the problem with two mutually excluded classes. A network with only one output neuron can be used. In this case, we try to use output neurons with softmax output functions. It will facilitate computation of weight updates as they can be calculated according to the classical back-propagation scheme. [50]

We’ve decided to choose the softmax output activation function and two binary neurons, one for each category categories. The output layer consists of two neurons. There are two groups that input items are assigned into. These groups are mapped on values of output parameter called extended work on Use-Case, which can take value 0 (false) or value 1 (true). For output layer, we’ve decided to use the softmax activation function (3.11). The softmax activation function is popular as activation function for neural networks. It converts an arbitrary real-valued vector into a multinomial probability vector. It’s used in classification problems. This is version of winner-take-all nonlinearity, in which maximum output is transformed to 1.0. [51]

\[
h(z) = \frac{e^z}{\sum_{i=1}^{n} e^{z_i}}
\]

(3.11)
Mean Squared Error

For neural network applied in following experiments Mean squared error (MSE) was used, because the output targets are discrete. The definition for the MSE is given in the following equation (3.12), where \( n \) is the number of patterns in the validation set, \( m \) is the number of components in the output vector, \( o \) is the output of a single neuron \( j \), \( t \) is the target for the single neuron \( j \), and each input pattern is denoted by vector \( i \).

\[
\text{MSE} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} (o_{ij} - t_{ij})^2}{n} \tag{3.12}
\]

Supervised training of the neural network

The two most useful training procedures are batch and online. The batch (offline) training means that the input data are entered into the network sequentially. After all input vectors are processed the total MSE equation (3.12) value is computed by comparing the network outputs with the real target values. The network parameters weights and biases are than updated according to this error value obtained.

Online training is the process of calculation of weight change for each individual training example. Therefore, it can be assumed that the error will decrease as the training examples increase [57]. In next paragraphs, training of the feed-forward neural network using back-propagation online and batch (offline) training algorithm is described.

The feed-forward neural network was trained using the back-propagation algorithm, which consists of four steps. As each item of training data is presented to the neural network, the error is calculated between the actual output of the neural network and the output that was expected. The weights are then modified, so there is a greater chance of the network returning the correct result in next epochs of the training. The Update-Weights method updates the weights and biases, with desired values (eta – learning rate, alpha - momentum), assumes that method for setting weights and method for computing outputs have been called. This method for update weights using back-propagation algorithm, which we have implemented in our application, involves four following steps:

1. In the first step, where we are computing output gradients (gradients for each output-layer node) we are using derivative of softmax function:

\[
\text{Softmax}' = (1 - y) \ast y \tag{3.13}
\]

Gradients are a measure of how far off, and in what direction (positive or negative) the current actual neural network output values are, compared to desired.
2. Usage of the output gradient values to compute gradients for each hidden-layer node. Hidden node gradients are computed differently from the output node gradients. We are using derivative of hyperbolic tangent activation function:

\[ \text{Tanh}' = (1 - y) \times (1 + y) \] 

3. Usage of the hidden node gradient values to compute a delta value to be added to input-to-hidden weight. Use hidden-node gradient values to compute a delta value for each input-to-hidden bias value.

4. Usage of the output-layer node gradients to compute a delta value for each hidden-to-output weight value. Use the output-layer node gradients to compute a delta value for each hidden-to-output bias value.

3.4. Experiment – Classification of Use Cases

The goal is to choose right technique for support of software effort estimation need to be chosen. This choice is possible to do after the experimental work and evaluation of results. Several techniques for classification of use cases were applied. Some of previously mentioned are e.g. SOM and SVM approaches. Both of them are explained in papers [29, 30] published by research group of software engineering. Two new techniques were suggested. The first one is feed-forward neural network with backpropagation training algorithm and second is statistical method called Naïve Bayes classifier. Both of them require some data pre-processing and also specific approach for processing data. Whole experiment is detailed described in following chapters.

4.5.1 Parameterization of Use Cases

The use case model of the evaluated project includes a set of parameters for each use case. Basically, three types of parameters are used: descriptive, structural and really evaluated parameters. These parameters are based on the use case point (UCP) algorithmic model. [19]

a) Descriptive parameters are evaluated from the description of the use case scenario, those parameters are explained in detail in papers [29, 30, 58]. We use following descriptive parameters: Overall difficulty is degree of complexity derived from number of words, rows and paragraphs within the use case. RFC means identifier of the project.
b) **Structural parameters** are evaluated by the structural or relational property of the use case on a given project. Set of structural parameters was defined as a result of the interview with several senior project managers. Following structural parameters and values are used:

NSC is short for type of implemented functionality of the system. It can have one of values New = 3, Standard = 1, or Change = 2. Next structural parameter is called Concerned activities (values of parameter” 1, 2, 3) which means, how many business processes are touched by the implementation of this use case. For example, 1 – summary use case or 0 – user or sub function. One of factors that can influence overlapping of time is homeworking. The parameter that specifies is some functionality was implemented on site or no, is called Implementation remote. This parameter can take value 1 – work can be done remote, or 0 – work must be done onsite. **Last of structural parameters is Testing level. This parameter takes a value** Easy = 0, Normal = 1, or Complex = 2. It means how difficult will be the testing of implemented functionality.

c) **Evaluated parameter** is evaluated backward after project end. This parameter means working time of developers. Overlapped working time is called “extended work” in introduced vocabulary. The parameter is set up for each particular use case. Use case scenarios are used in requirements, analysis and design stages in the software life cycle. Extra work can be in two states: 1 - additional work turned up, and 0 – without additional work. If there is some additional work then was expected for the use case scenario. [4]

Figure 5: Example of the parameterization and transformation of use case scenario.
Data Pre-processing

The performance and speed of classification algorithms depends on quality of a dataset. Low-quality training data may lead to overfitting of classifiers. Data pre-processing techniques are needed, where the training data are chose for classification. Pre-processing of data can improve the quality of them and also it can help to improve the accuracy results. There is a large a number of different data pre-processing techniques. Data cleaning and reduction have been used in that case. Data cleaning means removal of noisy data. Data reduction, it is reducing the data size by aggregating and eliminating redundant features.

As a part of the cleaning process, columns that contain same values for all rows have been removed. Parameter called “extra work” was divided into 2-values binary vector. It is required for the classification using softmax output activation function. The training matrix includes 9 columns for input parameters plus 2 extra work columns.

Furthermore, parameters: Use Case type, Work remote and Implementation remote were excluded from the training and testing process because the have equal values for each row. List of parameters for training and testing matrix after data pre-processing includes: Dataset id Dif. Rows, Dif. Paragraph, Dif. Words, Overall dif., RFC, N/S/C, Concerned activities, Testing level, XWorkYes, XWorkNo. You can see the example of parameterization and transformation of use case scenario in following picture.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty rows</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Difficulty paragraphs</td>
<td>Easy</td>
<td>Complex</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Difficulty words</td>
<td>Easy</td>
<td>Medium</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>Difficulty overall</td>
<td>Easy</td>
<td>Medium</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>RFC</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
</tr>
<tr>
<td>N/S/C</td>
<td>Complex</td>
<td>Medium</td>
<td>Easy</td>
<td>Medium</td>
</tr>
<tr>
<td>Concerned activities</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>UC type</td>
<td>Subfunction</td>
<td>Subfunction</td>
<td>Subfunction</td>
<td>Subfunction</td>
</tr>
<tr>
<td>Work remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
</tr>
<tr>
<td>Implementation remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
<td>Remote</td>
</tr>
<tr>
<td>Testing level</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Extended work</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: Example of parameterized use cases.
An example of four use cases is showed set in Table 1. Two different groups of use cases have been chosen. The first group includes use cases with value “No” of extended work parameter (#16 and #1023). Second group includes use cases with value “Yes” of extended work parameter (#932 and #1026). Probably you caught that parameters use case type, work remote and implementation remote are same for all items. Actually they are same for whole dataset. That is the reason, why these parameters are not important, so they can be removed.

**Dataset**

The whole datasets consists of 1041 Use Cases from years 2008-2013. These items are divided into 6 datasets. Each dataset includes last 10 testing items. The number of training items in dataset for year 2008 is 385, for year 2009 is 569, for year 2010 is 634, for year 2011 is 738, for year 2012 is 953 and for year 2013 is 1041 items. We use three sets of parameters: descriptive, structural and really evaluated parameters. [58]

There are two approaches for testing. The first kind of test is using last 10 items of the current dataset. These items are excluded from the training set and included to the testing set. Data sub-sets of particular years are subsequently added into the neural network in six iterations. Use cases were divided into two categories. In the first category extra work on use case was needed. In this case $x_{Work}$ vector obtain values $[1, 0]$. If extra-work was not needed than vector has values $[0, 1]$.

### 4.5.2 Classification using Neural Network

**Training of Neural Network Classifier**

The first step called „Train Neural Network“ consists of six cycles within data set groups are continuously executed. In the first cycle, reading algorithm reads the first data set and starts the training of the neural network. When the training had finished and we had received the result of testing of the neural network the second cycle can start. In a second cycle, the reading algorithm reads the first and second data set and starts training of the neural network using both of training data sets (the first and second data set). Third cycle uses the data set 1-3, etc. „Test Neural Network“ is the step, within last 10 items of the data set are always excluded from the training data set and form the testing data set. „Add Next Data Set“ means that data sub-sets of particular years are subsequently added into the neural network training data set in six iterations.
The last step of whole process is called “Evaluation”. During this step, use cases were divided into two categories. In the first category, some extra work on use case was needed. In that case vector - xWork has two values [1,0]. If extra-work was not needed than vector has values [0,1]. States of particular use case are described by state chart diagram in Figure 7.

Figure 6: Overview of the classification process using neural network.
Figure 7: State-chart diagram describes states of Use Case after classification process.

The main purpose of classification of use case is a prediction of value of parameter called $xWork$ (extended work) for support of effort estimation. This $xWork$ parameter is important for identification of risk of underestimation. This section contains information about the logic of the method for estimations. The method is focused on estimation of future parameter of the use case, based on the known parameters in the beginning of the project.

Feed-forward neural network classifier for estimation of future parameter has been used. This estimation is based on results of classification process. The preliminary thing of our approach is that the use cases are written in standardized way. That means if we would like to describe something in the use case we always use the same type of sequences like before. And another preliminary thing is that we parameterize particular use cases. First type is descriptive parameters that are automatically computed according the use case style and second type are structural parameter that is filled by analyst that creates current use case. [14]

Evaluation - Results of Classification using Neural Network

The neural network is trained using back-propagation online and batch (offline) training algorithms with data set of 1041 use case scenarios. Use case scenarios are parameterized and transformed into vectors of double values. The binary (two-elements) vectors are used for solving classification problem with two classes.

Following Table 2 shows the best-measured results of testing neural network after online training process. The training accuracy is computed using training data set, which includes all vectors from current data set and excludes last 10 vectors. The testing accuracy is computed on testing data set, which includes last 10 vectors from current dataset and excludes them from training dataset. [14]
<table>
<thead>
<tr>
<th>Use Case sets from years</th>
<th>Training iterations</th>
<th>Hidden neurons</th>
<th>Training accuracy / testing accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>5832</td>
<td>3</td>
<td>71.03% / 20%</td>
</tr>
<tr>
<td>2008-2009</td>
<td>2744</td>
<td>5</td>
<td>72.16% / 80%</td>
</tr>
<tr>
<td>2008-2010</td>
<td>5832</td>
<td>3</td>
<td>73.36% / 90%</td>
</tr>
<tr>
<td>2008-2011</td>
<td>2744</td>
<td>5</td>
<td>73.98% / 100%</td>
</tr>
<tr>
<td>2008-2012</td>
<td>1728</td>
<td>6</td>
<td>74.33% / 70%</td>
</tr>
<tr>
<td>2008-2013</td>
<td>1000</td>
<td>7</td>
<td>75.51% / 90%</td>
</tr>
</tbody>
</table>

Table 2: Results of online training.

Table 3 shows the best-measured results of testing neural network after batch (offline) training process. The accuracy of testing is also computed on testing data, which includes last ten items (vectors) from current dataset.

<table>
<thead>
<tr>
<th>Use Case sets from years</th>
<th>Training iterations</th>
<th>Hidden neurons</th>
<th>Training accuracy / testing accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>2744</td>
<td>5</td>
<td>70.85% / 20%</td>
</tr>
<tr>
<td>2008-2009</td>
<td>8000</td>
<td>2</td>
<td>71.85% / 100%</td>
</tr>
<tr>
<td>2008-2010</td>
<td>4096</td>
<td>4</td>
<td>73.37% / 90%</td>
</tr>
<tr>
<td>2008-2011</td>
<td>8000</td>
<td>2</td>
<td>73.75% / 100%</td>
</tr>
<tr>
<td>2008-2012</td>
<td>4096</td>
<td>4</td>
<td>74.33% / 70%</td>
</tr>
<tr>
<td>2008-2013</td>
<td>8000</td>
<td>2</td>
<td>75.60% / 90%</td>
</tr>
</tbody>
</table>

Table 3: Results of batch training.

Next, neural network is tested using all vectors from current and all vectors from next dataset. For example, the neural network is trained using use cases from year 2008 and is tested using use cases from year 2009. Table 4 shows results of testing neural network after online training process. As testing items we used all use cases from next year dataset.
Table 4: Results of online training using testing set from next year.

Table 5 shows the best-measured results of testing neural network after batch training process. As testing items we also used all use cases from next year dataset.

Table 5: Results of batch training using testing set from text year.

The training accuracy is computed using training data set, which includes all vectors from current data set and excludes last 10 vectors. The testing accuracy is computed on testing data set, which includes last 10 vectors from current dataset and excludes them from training dataset. You can see results in Table 5. Trained neural network was tested using all vectors from current and all vectors from next dataset. For example, the neural network is trained using use cases from year 2008 and is tested using use cases from year 2009.
4.5.3 Classification using Naïve Bayes

Naïve Bayes is a simple and very powerful technique that you should be using on Use Case classification problems. It is also kind of supervised training method.

Process Overview

The whole process consists of five important activities. The first is Data Preprocessing, second is training, and third is testing type selection, which allows using selected vectors by project manager or all vectors from the next dataset for testing purpose. At the end of the process is activity called Evaluation. [59]

Figure 8: Overview of the classification process using Naïve Bayes Classifier
**Train Naïve Bayes Classifier**

Train the classifier means that we need to tell to classifier, that the given features resulted in the given category (extended work is Yes or No). These items are divided into 6 datasets. Each dataset includes last 10 testing items. Data sub-sets of particular years are subsequently added into the neural network training data set in six iterations. After the training process, we would like to determine which posterior is greater, extended work “No” or extended work “Yes”. For the classification as No group the posterior example is given using formula (3.3):

\[
posterior\ (No) = \frac{P\ (No) \times P\ (difficultyRows|No) \times P\ (difficultyParagraphs|No) \times \ldots}{evidence}
\]

For the classification as Yes group the posterior example is given by similar way, using (3.3):

\[
posterior\ (Yes) = \frac{P\ (Yes) \times P\ (difficultyRows|Yes) \times P\ (difficultyParagraphs|Yes) \times \ldots}{evidence}
\]

The evidence example, which may be calculated like this:

\[
evidence = P\ (Yes) \times P\ (difficultyRows|Yes) \times P\ (difficultyParagraphs|Yes) \times \ldots
+ P\ (No) \times P\ (difficultyRows|No) \times P\ (difficultyParagraphs|No) \times \ldots
\]

**Test classifier using 10 Vectors from the Current Dataset**

The algorithm for measurement of accuracy computes the percentage of correct classifications. This algorithm for computing accuracy uses a winner-takes-all approach. Test-data set includes 10 items from current dataset selected by project manager.

**Test classifier using all Vectors from the Next Dataset**

Finally, the test-data set includes all items from the next set. This data set is provided to the same algorithm and accuracy is computed.

**Evaluation - Results of Classification using Naïve Bayes**

Use cases were divided into two categories. In the first category extra work on use case was needed. In this case xWork parameter obtain value 0. If extra-work was not needed than parameter obtain value 1. Table 6 shows results of testing process using 10 testing vectors selected by project manager from current dataset as “testing accuracy” and all training vectors also from current dataset as “training accuracy.”
Use Case sets from years | Training accuracy / testing accuracy
--- | ---
2008 | 63.28% / 40%
2008-2009 | 66.41% / 60%
2008-2010 | 65.78% / 70%
2008-2011 | 66.20% / 90%
2008-2012 | 66.91% / 70%
2008-2013 | 68.01% / 60%

Table 6: Results of training using testing set of 10 use cases.

Table 7 shows results of testing process using 10 testing vectors selected by project manager from the next dataset as “testing accuracy” and all training vectors also from the next dataset as “training accuracy.”

<table>
<thead>
<tr>
<th>Use Case sets from years</th>
<th>Training accuracy / testing accuracy</th>
<th>Testing accuracy (all vectors / 10 vectors) from year:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>63.28% / 40%</td>
<td>2009 (102 vectors) 63.76% / 70%</td>
</tr>
<tr>
<td>2008-2009</td>
<td>66.41% / 60%</td>
<td>2010 (110 vectors) 66.71% / 80%</td>
</tr>
<tr>
<td>2008-2010</td>
<td>65.78% / 70%</td>
<td>2011 (110 vectors) 66.43% / 90%</td>
</tr>
<tr>
<td>2008-2011</td>
<td>66.20% / 90%</td>
<td>2012 (110 vectors) 66.60% / 70%</td>
</tr>
<tr>
<td>2008-2012</td>
<td>66.91% / 70%</td>
<td>2013 (109 vectors) 66.94% / 70%</td>
</tr>
<tr>
<td>2008-2013</td>
<td>68.01% / 60%</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Results of training using testing set of 10 use cases.

This experiment is focused on usage of Naive Bayes Classifier in field of effort estimation support. Classifier is used then for the classification of the use cases in our supportive method for software project estimation. The results show that the method can estimate an extra work parameter with the probability for testing use cases selected by manager in most cases between 60-90% success. Detailed results of experiments based on datasets from years 2008-2009, 2008-2010, 2008-2011 are showed in Table 6 and Table 7. As we can see, the accuracy is higher when there are more data available for the training. The first dataset was trained with more than 60% accuracy but the testing set showed 40% accuracy using Naive Bayes. These results seem to be promising for our purposes so far. Anyway, more experiments have
to be done to demonstrate the general applicability even for our purposes. Future work will be focused on experiments using other statistical method called logistic regression and comparison with provided Naïve Bayes based approach. [59]

4.5.4 Summary of Results

The different between accuracy of testing after trained feed-forward neural network and Naive Bayes Classifier is shown in Table 8 and Table 9. Results of testing with usage of last ten use cases from current dataset are showed in Table 8.

Results of testing using all use cases from the next dataset are showed in Table 9. For both approaches (feed-forward Neural Network and Naive Bayes), the classifier is trained using five datasets (2008 – 2012) and tested using also five datasets (2009-2013). It means that we used five datasets from six available datasets of Use Cases for training and testing. Results are copied from tables (Table 2, Table 3, Table 4, Table 5) for feed-forward neural network and from tables (Table 6, Table 7) for Naive Bayes Classifier.

<table>
<thead>
<tr>
<th>Training vectors from years</th>
<th>Feed-Forward Neural Network – Online Training (training / testing accuracy)</th>
<th>Feed-Forward Neural Network – Batch (Offline) Training (training / testing accuracy)</th>
<th>Naive Bayes Classifier (training / testing accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>71,03% / 20%</td>
<td>70,85% / 20%</td>
<td>63,28% / 40%</td>
</tr>
<tr>
<td>2008-2009</td>
<td>72,16% / 80%</td>
<td>71,85% / 100%</td>
<td>66,41% / 60%</td>
</tr>
<tr>
<td>2008-2010</td>
<td>73,36% / 90%</td>
<td>73,37% / 90%</td>
<td>65,78% / 70%</td>
</tr>
<tr>
<td>2008-2011</td>
<td>73,98% / 100%</td>
<td>73,75% / 100%</td>
<td>66,20% / 90%</td>
</tr>
<tr>
<td>2008-2012</td>
<td>74,33% / 70%</td>
<td>74,33% / 70%</td>
<td>66,91% / 70%</td>
</tr>
<tr>
<td>2008-2013</td>
<td>75,51% / 90%</td>
<td>75,60% / 90%</td>
<td>68,01% / 60%</td>
</tr>
</tbody>
</table>

Table 8: Results of testing using set of 10 testing use cases from the current dataset.
**Table 9:** Results of testing using set of all use cases from the next dataset.

Results show that neural network is appropriate classifier as a supportive technique for estimation of extra-work parameter. Test items (use cases) have been selected by project manager. Accuracy is for most cases between 70 and 100%. Data for performed experiment are from years 2008 – 2011 showed in Table 8 and Table 9.

The accuracy is higher when there are more data available for the training. The Naive Bayes classifier is trained with more than 70% accuracy. The testing showed 80% accuracy using feed-forward neural network. The issue is that project manager has selected training datasets. It means that he selected “the most problematic” use cases that he would like to have predicted. Anyway, the accuracy of classifiers is between 70 and 100% for the training data sets 2-6 for feed-forward neural network and from 60-70% for Bayes Classifier. The experiment shows that the feed-forward neural network provides results that are better in some cases then the results of experiments using Naive Bayes Classifier. These results seem to be promising for our purposes so far.

The main advantage of Neural Network based classification approach is higher accuracy. Disadvantage of this approach is longer training time. On the other hand, advantage of Naive Bayes Classifier is simplicity and shorter training time, but accuracy is lower than using neural network (NN).

These datasets include training data and testing data. Certain values are written in the previous Table 9. Provided chart shows that most accurate results of classification of software requirements in form of use cases are provided by neural network with feed-forward architecture and back-propagation training algorithm. The feed-forward neural network with back-propagation training algorithm has been selected as the most appropriate kind of classifier for next utilization in field of prediction using software requirements. [14], [59]
4. Exploratory Analysis of Software Requirements

In a previous chapter experiments that are using data in form of use cases were presented. As was mentioned before the software company has provided data for research and experiments. The data were gathered from period of six years of software development.

Another software company has provided another data from their project management database. This new data have to be analysed. Exploratory analysis of provided dataset is important step before using Machine-Learning algorithms or Artificial Intelligence Algorithms [61]. The main idea of this part of research within effort estimation field is to identify important parameters for future estimations, remove noisy parameters, clean, transform, normalize dataset and also create data model for future prediction. Present study has used knowledge based on historical data obtained from an existing software company.

Dataset is presented in form of informal software requirements with certain parameters. Software requirements were described during the first phase of software development – called “elaboration phase” [31]. Neural network algorithms for classification (prediction) are usually able to process quantitative or binary data - it is the reason, why is important to transform categorical data to binary. Next sub-section describes the dataset, parameters of items in dataset and statistical properties as well. Application of statistics is described in the following Section 4.2. The following section also explains the experimental approaches.

Finally, the closing Section 4.3 of exploratory analysis part shows results of the experiment and its visualizations using component plane (heat maps). It also provides an overall view to solution of “exploratory analysis using statistics and SOM” problem. “The understanding of a complex data requires consideration of a many statistical indicators describing its different aspects and their relationships.

The main point of exploratory data analysis is to present a data model in easily understandable shape to the original data model. [62] In this exploratory analysis, the Kohonen’s Neural Network for exploratory data analysis is used. “Kohonen’s Self-Organizing Map is a unique method that combines the goals of projection and clustering algorithms.” [62] The purpose of this analysis is to explore parameters of Software Requirement Entity. Then is needed to describe the influence of these parameters to each other as well as create appropriate data model for future processing using Machine-Learning methods and Artificial Intelligence [14]. [63]

4.1. Parameterization of Software Requirements

Requirements engineering process consists of several activities. Important activity of RUP process is called “Manage the scope of the system” and it is appropriate activity for effort estimation. The functional and non-functional requirements are collected and prioritized. The
system can be delivered on expected time and within the budget [10, 13]. Parameters showed in Table 10 can be divided into two groups:

- The first group is called “text parameters”.
- Second group is called “numerical parameters” – that includes Binary, Nominal or Quantitative values.

As it’s mentioned above, the Euclidian distance is used. Usage of this kind of distance measurement requires preparation of vector of quantititative double or binary values. Nominal variables can be also converted to binary variables, but there are only two nominal variables that are not so significant. The first variable is Category and the second one is State. Requirements with State “done” have been chosen, thus there is no reason to convert and use this variable (“State”) later. [63]

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Example value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>REQ-901</td>
<td>Text</td>
</tr>
<tr>
<td>Name (Name_length)</td>
<td>Implement login to CRM app</td>
<td>Text</td>
</tr>
<tr>
<td>Description (Desc_length)</td>
<td>Description of requirement…</td>
<td>Text</td>
</tr>
<tr>
<td>Category (Req. type)</td>
<td>Implementation of new feature</td>
<td>Nominal</td>
</tr>
<tr>
<td>Status</td>
<td>Done</td>
<td>Nominal</td>
</tr>
<tr>
<td>Sum of Estimated Hours (Sum_estim)</td>
<td>16</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Sum of Actual Hours (Sum_actual)</td>
<td>18.5</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Estimated Hours of Testing (Estim_test)</td>
<td>2</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Actual Hours of Testing</td>
<td>1</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Estimated Hours of Analysis (Estim_anal)</td>
<td>3</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Actual Hours of Analysis</td>
<td>3</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Developer productivity (Productivity)</td>
<td>1.2</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Priority of requirement (Priority)</td>
<td>0.9</td>
<td>Quantitative</td>
</tr>
</tbody>
</table>

Table 10: Example of requirement entity, including parameters with example values

Pre-processing improves data quality as well as the accuracy of the Machine-Learning algorithms applied to selected data. Several data pre-processing techniques have been used such as data cleaning, normalization, and transformation.

Cleaning is process of removing noisy data or parameters. The reducing of data size by aggregating and eliminating redundant features is necessary as well [43]. “Normalization is a "scaling down" transformation of the parameters. Within a parameter can be difference
between the maximum and minimum values, e.g. 0.01 and 1000.” [61] It needs to be scale
down to low values because Euclidian distance for distance measure is used.

Future exploratory analysis using Kohonen’s Self-Organizing Map requires selection of
quantitative and normalized parameters.[62]. Whole dataset consists of 1553 items after
filtering and cleaning process. [63]

4.2. Exploratory Analysis

In order to investigate distribution of variables, box and whisker plots were created. Box and
whisker plots visualize the basic distribution of parameters in given dataset in the category
indicating the median, first to fourth quartile, minimum and maximum as well as outliers –
abnormal observations in dataset.

It can show weather a dataset is symmetric (the median is roughly in the centre of the
box) or skewed (the median cuts the box into two unequal pieces). Variability in dataset –
described by five-number summary – is measured by interquartile range (IQR) which is equal
to Q3 – Q1 (the difference between the 75th percentile and the 25th percentile). Larger IQR
indicate that the data set is more variable [64].

First, the box and whisker plots were created for non-normalized data of variables,
which have the same unit. Lets see it in the Figure 9. The highest value distribution and the
highest maximum showed variable Sum_estim. Variable Diff_sum showed mostly minus
values. The lowest data distribution was observed for the variables Estim_impl, Estim_test,
Estim_analysis. All selected variables showed too many outliers. [63]

Figure 9: Box-plot and Whisker-plot of variables with same units (hours) - R studio
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Minimum</th>
<th>Mean</th>
<th>Median</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estim_impl</td>
<td>0.000</td>
<td>1.216</td>
<td>0.000</td>
<td>150.000</td>
</tr>
<tr>
<td>Estim_test</td>
<td>0.000</td>
<td>1.201</td>
<td>0.000</td>
<td>52.000</td>
</tr>
<tr>
<td>Estim_analysis</td>
<td>0.000</td>
<td>0.9676</td>
<td>0.000</td>
<td>80.000</td>
</tr>
<tr>
<td>Sum_estim</td>
<td>0.150</td>
<td>11.73</td>
<td>4.000</td>
<td>159.000</td>
</tr>
<tr>
<td>Sum_actual</td>
<td>0.250</td>
<td>7.426</td>
<td>2.750</td>
<td>186.250</td>
</tr>
<tr>
<td>Diff_sum</td>
<td>141.000</td>
<td>4.302</td>
<td>0.500</td>
<td>146.250</td>
</tr>
</tbody>
</table>

Table 11: Table of values with same unit (hours) from Figure 9.

Next step was normalization of dataset. The data are normalized in range from 0 to 1. For the comparison, box and whisker plots were created as well. (Figure 10) – however, in this case, plots were created for all of variables.

Data distributions varied for each variable. The highest data variability showed variables Name_Length and Priority. Moreover, variable Priority showed the highest data values and abnormal values were not observed. In the case of all remaining parameters, too many outliers were observed. Variables Desc_Lenght, Sum_actual, Sum_estim, Estim_anal, Estim_test, Estim_impl and Diff_sum had median values at the same level.

Figure 10: Box-plot and Whisker-plot of normalized variables - output R studio.
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Minimum</th>
<th>Mean</th>
<th>Median</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estim_impl</td>
<td>0.000</td>
<td>0.008107</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Estim_test</td>
<td>0.000</td>
<td>0.02309</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Estim_analysis</td>
<td>0.000</td>
<td>0.01209</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Sum_estim</td>
<td>0.0009434</td>
<td>0.0737601</td>
<td>0.0251570</td>
<td>1.000</td>
</tr>
<tr>
<td>Sum_actual</td>
<td>0.001342</td>
<td>0.039872</td>
<td>0.014765</td>
<td>1.000</td>
</tr>
<tr>
<td>Diff_sum</td>
<td>-0.964100</td>
<td>-0.029413</td>
<td>-0.003419</td>
<td>1.000</td>
</tr>
<tr>
<td>Productivity</td>
<td>0.2083</td>
<td>0.6884</td>
<td>0.6917</td>
<td>1.000</td>
</tr>
<tr>
<td>Priority</td>
<td>0.5882</td>
<td>0.8325</td>
<td>0.7059</td>
<td>1.000</td>
</tr>
<tr>
<td>Desc_length</td>
<td>0.00022</td>
<td>0.04127</td>
<td>0.02555</td>
<td>1.000</td>
</tr>
<tr>
<td>Name_length</td>
<td>0.0400</td>
<td>0.3709</td>
<td>0.3400</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Table 12: Table of normalized values from Figure 10.

In the next step, the Principal Component Analysis (PCA) was performed in the R project environment [65] using the FactoMineR package [66]. “PCA represents statistical method for reducing the dimensionality of a data set into a smaller dimensional subspace prior to running a machine-learning algorithm.” [67] The correlation coefficient is expressed as a cosine of the angle that two variables in the model. (Figure 11) [67] Relationships between Diff_sum and other variables were investigated. [63]

Figure 11: Correlation circle of PCA
The correlation circle indicated strong positive correlation between group of variables *Priority, Name_length* and *Desc_length* and also between group of variables *Estim_impl, Estim_anal* and *Estim_test*. A strong negative correlation was observed between *Productivity and Priority, Productivity* and *Name_length; Diff_sum and Estim_test, Estim_anal*. The Pearson correlation coefficient was also calculated between all variables and the correlation matrix (Figure 12) was performed in R programme as well, using the “corrplot Package”. The highest values of correlation coefficient were observed between variables *Diff_sum* and *Sum_estim* (negative correlation) and *Sum_actual* and *Sum_estim* (positive correlation). [63]

![Correlation Matrix](image.png)

Figure 12: Correlation matrix
Exploratory Analysis using SOM

“The Self-Organizing Map (SOM) is an adaptive display method particularly suitable for representation of structured statistical data.” [62] The SOM is also called Kohonen’s network and it is kind of unsupervised neural network algorithm developed by Teuvo Kohonen [68].

Topology of SOM

Figure 13 shows that hexagonal topology of SOM output layer is selected. Each cell of presented topology represents one of output neuron in Kohonen’s Map. Numbers inside neurons shows amount of input items assigned to certain neuron. For instance, neuron located in left-bottom corner includes 2 input (training) items – This number is called “hints” in MATLAB environment, which is used for this experiment.

Figure 13: Hexagonal topology of SOM (25 × 25) constructed by MATLAB environment.
**SOM Algorithm**

SOM algorithm provides clustering and assigning input items into the clusters (represented by neurons of Kohonen’s output layer). [69] For better description of the original SOM algorithm, let define \( X \) that represents input vectors \( x \) in step (time) \( t \) as \( \{x(t)\} \). \( M_i \) represents vectors \( \{m_i(t)\} \) that computed approximations of the model \( m_i \). Letter \( i \) is in index of neuron (node/cell) with which \( m_i \) is associated [70]. Main formula (4.1) for computing weights of output Kohonen’s layer:

\[
m_i(t + 1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)], \tag{4.1}
\]

where \( h_{ci}(t) \) is called neighbourhood function. This function has influence on change of weight of neurons in neighbourhood with winner neuron. The neighbourhood function plays important role in self-organization of the map. In this case, the Gaussian neighbourhood function was used.

The main goal of each cycle of the training process is to find winner neuron. Winner is the nearest neuron to provided input. Distance between neuron and input is measured using Euclidian Distance (ED). Letter \( c \) is index of neuron with minimum ED (winner neuron) it is the nearest neuron \( m_c \) to provided input \( x_c \):

\[
c = \text{arg} \min_i \{||x(t) - m_i(t)||\}, \tag{4.2}
\]

**Configuration Initialization**

The goal of this approach is to identify the clusters in a dataset and also visualize distribution of the single variables. The output space of the neural network shows groups of software requirements clustered using SOM with Euclidian distance and the most common – Gaussian neighbourhood function. Important parameter of this Gaussian neighbourhood (GN) function is variable radius. In this case – the radius is 25. If \( Ud \) is the abbreviation for the squared-radius from the winner neuron, \( \exp \) is the abbreviation for the exponential function, and if, for brevity, we write \( \text{radius}(t) \), then the Gaussian neighbourhood function GN is written as: [70]

\[
GN = \exp(-\frac{Ud}{2\text{radius}(t)}) \tag{4.3}
\]

A value of parameter decreases linearly with training steps. The number of training iterations also decreases during the training. Initial value was set up to 5000. Next important parameter of this layer is “lattice” with value of “hexagonal”. As is showed in the Figure 13, there is (25 x 25) 625 neurons within output layer. Neurons of Kohonen’s layer are initialized by small random values greater than zero.
The component plane has been used to visualize distribution of each single variable of our software requirements dataset in Figure 14 in following Section 4.3.

4.3. Visualization of Results of the Clustering

Exploratory analysis and clustering help us to find important parameters and values those are acceptable for future usage in methodology for effort estimation. In that section the visualization of distribution of selected variables is provided in component plane (Figure 14).

Provided component plane provides the graphical output of clustering using Self-Organizing Map (SOM). SOM is a method for data-analysis that shows similarity and relations in a set of data. [70] The component plane shows neurons of Kohonen’s (output) layer and values of that layer (a.k.a. Kohonen’s layer). Each square in following figure (Figure 14) shows the weights from the provided input to the layer's (output) neurons. [63]

![Figure 14: Component plane of Kohonen’s layer of SOM](image)
Component plane depicts distribution of variables, in order: length of Name of requirement, length of Description of requirement, Priority of requirement, Sum of actual hours – normalized working time, Sum of estimated hours, Estimated hours of analysis, Estimated Hours of test, Estimated hours of implementation, Productivity of employee, and type of requirement (New / Bug / Update), and Requirement is underestimated.

Distribution of variables in component plane (Figure 14) shows that the highest value of variable Name length is usually in places with high value of variable Priority, and partially in places with highest value of variable Req. type “New”. In other words, software requirement with long name has usually high priority and it is type (category) - “New”, e.g. implementation of new functionality as is written in example (Table 10). Component plane also shows that requirements with higher estimations have also high priority. Very important parameter is Req. is underestimated – this is binary parameter computed by subtracting Sum actual from Sum estimated (positive result has value 0, negative result has value 1). Using this parameter Req. is underestimated it is clear that underestimated requirements are always type of “Bug” or “Upd” (update), partially also type of “New”. Underestimated requirements usually have high priority. Other parameters, e.g. productivity of developer have no significant influence on value underestimation - In case where analyst makes estimation taking account productivity coefficient of particular developers.

4.4. Results Summary of Exploratory Data Analysis

Statistical procedure called Principal Component Analysis (PCA) shows relations between variables. In summary, strong positive correlation between requirement-parameters: priority, length of requirement name, and length of requirement description says that requirements with high priority have usually long name and long description. Strong negative correlation between productivity and priority says that requirements with high priority are usually assigned to developers with low productivity coefficient (low productivity coefficient means – skilled, senior developer). High values of correlation coefficient were observed also between variables Diff_sum and Sum_estim (negative correlation) and Sum_actual and Sum_estim (positive correlation) in the Figure 11.

In addition, Kohonen’s Self-Organizing Map provides output of clustering in form of component plane. This component plane (Figure 14) shows that requirements with high priority are usually type of “New” and has longer name. It also shows, underestimated requirements are always type of “Bug” or “Upd” (update) and they have also high priority. Figure 14 also shows inverse relation between variables “Diff sum” and “Sum estimated”.

Finally, the analysis of software requirements using both statistical and machine-learning method has been made. While statistics describes correlations between variables, on the other hand Kohonen’s (SOM) neural network provides another point of view on the same
data set. Results of performed exploratory analysis are important for future work in field effort estimation presented in this thesis. Mentioned work is presented in the next Section 5 and it’s focused on design of a methodology for effort estimation supported by machine-learning technique for classification, particularly multi-layered neural network. [63]
5. Proposed Method for Effort Estimation

The approach for software effort estimations based on historical data from database of application for project management in a real software company is proposed in this section. Project managers, analysts and developers, use this application every day during the development process. The application is usually used for evidence of software projects, requirements, tasks, employees and worksheets. The data collected into database on daily base allow to project managers or analysts to make more accurate predictions.

Estimation of Working-Time

Proposed approach is unique by usage of machine-learning techniques for prediction of working-time of particular tasks. These tasks are based on requirements, inserted to the project management application by project manager. Requirements include information like name of a project, type requirement, or requirement priority. It also includes estimations of time for consultations, time for requirements analysis and writing of documentation. We need to include also time for software project management, software development, support, test, deploying of database, deploying of implemented functionality, training of users, and finally sum of all estimations.

Each requirement includes name and text description. Requirements are divided into tasks that have to be assigned to specific employees of a software company. Each employee should be in role of developer, or technical support. The main reason for inaccurate estimation is a fact that there are many factors, which are affecting the accuracy of estimations done by estimator (project manager or analyst). An inaccurate prediction calls for usage of machine-learning supportive techniques, which should help to ensure higher accuracy of time estimations. The methodology proposed in this section is focused on estimation of time required for implementation of tasks by software developers. These tasks can be assigned to employee in a company with role developer or helpdesk programmer.

Estimation Process Overview

Particular activities of the effort estimation process are better described in activity diagram in Figure 15. The process starts with the first activity (“Data selection from SQL database“). Second activity („Data processing and Parameterization“) includes cleaning, removing duplicities, etc. Parameters can be divided into numerical and text. Numerical parameters should be parameterized (converted to integer value). For this purpose, there is implemented function that counts words inside the name and description fields and returns numerical value. This function returns converted items into numerical outputs. In other words, parameters with text values are replaced by parameters with integer values.
This collection of pre-processed data is saved to database, which is also known as the training database. Items included in the training database are utilized for training of neural network. The goal of a training process is to adjust weights of a neural network. When a training process of a neural network is completed, the project manager can use supportive tool that utilize the trained neural network. A graphical description of the effort estimation process supported by neural network is depicted in the Figure 15.

![Figure 15: Activity diagram – Overview of method form the top-level perspective.](image)

**Pre-condition for Deployment of Methodology**

The method is appropriate for a software company with deeper historical knowledge database, available of estimators and also with general historical data from a development of software products. Important activity is tracking of actual working time (estimated vs. actual time) during the whole software development process.

**Roles**

- **Project Manager / Estimator** – Person who is responsible for estimations time to develop the system and assigning tasks to developers.
- **Developer** – Person who implements desired functionality of the system and performs unit testing.

**Artefacts**
- **Historical Data** – Database of a company, which stores information about software requirements, developers and worksheets.
- **Training Set** – Pre-processed data that is ready to be used for training of machine-learning algorithm. In other words, a training set is a collection of items that include numerical quantitative parameters (usually double-value vectors).
- **Software Requirement** (functional) – Single item that contains stakeholder’s needs, specified by requirements specifier for software developer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Defines the main requirement objective. It is usually defined as a default goal of implementation of desired functionality.</td>
</tr>
<tr>
<td></td>
<td><em>Example “Login form to the administration section.”</em></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Particular task is described a little bit detailed.</td>
</tr>
<tr>
<td></td>
<td><em>Example “The login for administrators should be created. This form contains input label – user name and input label – password. Form also includes button – Login. User enters user name and password and clicks on the button. System verifies user name and password against database. Verification passes and user is redirected to administration section.”</em></td>
</tr>
<tr>
<td></td>
<td>Possible values are Yes or No.</td>
</tr>
</tbody>
</table>

**Homeworking (optional)**

**Type of Requirement**
- Categorical parameter for categories:
  - Implementation of new functionality
  - Update/change existing functionality
  - Fix a bug

**Status**
- Categorical parameter for categories:
• Created
• In progress
• Completed

**Priority**
Priority ratio is in range from 0.1 to 1. Value 1 represents most important requirements and value 0.1 represents requirements with lower importance.

**Developer Ratio**
Requirement has to be assigned to developer. Developer Ratio parameter that represents productivity of particular developer.

**Estimated Time - Analysis**

**Estimated Time - Implementation**

**Estimated Time – Test**

**Sum of Estimated Time**
Sum of estimated time = estimated time for analysis + time for implementation + time for test

**Actual Time**
Actual time is entered, after work on task ends. The task is state of completed.

Table 13: Artefact – Template for software requirement.
5.1. Data Model, Inputs and Outputs

Example of data model in the following Figure 16 is extracted from the real SQL database of software application for project management in the software company. This application stores information about software projects, developers, requirements, and software products.

This section provides selection and description of important information about entities, which includes parameters that influence time of development the software product or just particular functionality. Particular entities that include important parameters are: requirement, employee and task. Actual effort is included in the entity called worksheet.

Figure 16: Class diagram - Data model of example project management application.

An example of the data model provides important information for future estimations of working-time. Entities depicted in the Figure 16 are detailed described in following paragraphs:

**Employee**

This entity represents employee of company. Each employee is characterized by code, name and productivity. Productivity is double-value parameter that expresses seniority of current developer. Junior developer has productivity 1.5 and senior developer has productivity 0.85.
Product
The product entity is characterized by code, name and description. Products are concrete software ERP and CRM applications developed by company. Each requirement is assigned to the product.

Project
This entity represents a software project. It is characterized by name and code. The development of a project goes through a number of requirements, thus it consists of requirements. Each requirement is assigned to the project.

Requirement
Represents a single requirements specification of the system. It has parameters like code, name, description, category, type, status and each requirement can be assigned to project, product and employee.

Task
Each Task has code, name, description, and other parameters like category, estimated hours, real hours, and priority value and. Each task is an assigned employee and requirement. In other words requirements consists of number of tasks. Some tasks can be marked as children. It means that they are assigned to parent task, trough parent code.

Worksheet
Worksheet is important entity for summarization hours of work on each task assigned to developer. Using information from worksheet, we are able to compare estimated and real hours. Worksheet includes parameters like code, name, and description. Each worksheet has to be assigned to task and employee.

Inputs
The data model provided above shows us entities and parameters. Inputs are selected from this model are divided into numerical quantitative, numerical categorical and also text inputs. All of these inputs are known before estimation process. Proposed methodology requires following inputs:

- Name – text input
- Description – text input
- Priority – numerical quantitative input.
- Sum of estimated time – numerical quantitative input
- Estimated time for analysis – numerical quantitative input
- Estimated time for implementation – numerical quantitative input
- Estimated time for test – numerical quantitative input
- Productivity of assigned developer – numerical quantitative input
- Type of requirement – numerical categorical input

**Output**

The main point of this method is to provide time-estimation of work on certain software requirement. We can call it “Working time” of developer on some task defined by software requirement created by an analyst. This working time is predicted by classification of requirement into time-groups. The methodology, which is presented in this thesis, suggests usage of eight time-groups. The number of time-groups is custom value and can be optionally changes according to company needs.

The reason for providing feature of custom configuration is that each software company can have different environment, so possibility of configuration of output classes according to company needs was necessary step. Suggested number of time-groups depends on experiences of project managers, results of experiments and exploratory analysis that we have done in the previous Section 4.

The first time-group is appropriate for small tasks, described by requirements estimated between 1 and 2 hours. This group is appropriate for type of requirements like change in source code, change in database, and update existing code or some easy installation. Requirements included in that group can be type of support.

Next groups of tasks can are based on requirements estimated from 3 to 4 hours, from 5 to 8 hours, from 9 to 16 hours of working time. We can see that duration of estimation is increasing. Duration of work on tasks can be half day, one day, and two days, etc. from another look. During analysis and consultations with company experts was observed one unspoken rule. This rule says that, if duration of working time of particular task is higher, then precision of estimation is less accurate. In other words, developers use to overlap time of estimation if tasks have longer duration of estimated time of work.

Based on this information the one of best practices (in the Section 5.3) has been proposed. The goal of this best practice is to suggest to keep estimations of tasks as short as possible - it is also one of best practices provided as a part of the methodology. M. Jørgensen has published interesting experiment focused on this problem. His article has title “Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays”. For more information please check the reference - [71].
On the other hand, sometimes it’s not possible to keep estimations of time too short. In order to this, method introduces also other time groups with duration from 17 to 48 hours, from 49 to 96 hours, from 97 to 160 hours, and finally from 161 hours and more.

Actually, there is no point to estimate a time for more than two work weeks, which is equal to 80 hours. So the last group for time estimation should be from 49 to 96 hours, because actual time is influenced by a lot of reasons. One of things should be illness of developer, blackout in company, or some hardware reason.

Following Figure 17 shows us usage of methodology by estimator. Project manager or analysts (estimator) enters values for input parameters. Input parameters in the following Figure 17 have only illustration character. These inputs have to be pre-processed. Text values have to be converted into quantitative values, number values need to be normalized and then items can be classified. Results of this classification help to estimator make estimations more accurate. In other words, estimator is able do double-check each estimation.

![Input Parameters](image)

**Input Parameters**
- Name
- Description
- Priority
- Sum of estimated time
- Estimated time for analysis
- Estimated time for implementation
- Estimated time for test
- Productivity of assigned developer
- Type of requirement

![Pre-processing | Classification](image)

**Results**
- 0 - 2 hours
- 3 - 4 hours
- 5 - 8 hours
- 9 - 16 hours
- 17 - 48 hours
- 49 - 96 hours
- 81 - 160 hours
- 160 and more

Figure 17: Daily-usage of Methodology

5.2. Deployment, Configuration and Usage of Methodology

Effective usage of the methodology requires understanding of goals and proper configuration. Each company has little bit different environment, so the main goal was define pattern that can be adjusted by configuration of parameters. Figure 18 shows whole methodology lifecycle. This lifecycle goes through integrations and it consists of three important phases. The first phase is deployment and initial configuration of the methodology. Second is usage of the methodology and third phase is evaluation and improvements. Particular activities of these phases are better described in following paragraphs.
Figure 18: Lifecycle of Methodology

**Deployment and Initial Configuration**

The first phase that is called “Deployment and Initial Configuration” includes step - identification of requirements of certain software company. Person who is deploying that methodology should ask questions:

- What is current accuracy of estimations?
- How many parameters we can provide as an input?
- How many output classes do we expect?
Next step in this phase should help describe development team in the company. Each developer is specific with different speed of coding or different qualification level. We are dealing with situation when estimator (person who is responsible for estimations, e.g. project manager or analyst) knows developers who will work on estimated tasks. Third step includes descriptions of data loading. Fourth step describes necessary pre-processing for usage of machine-learning algorithms. These steps are detailed described in next paragraphs after activity diagram in Figure 19. Following activity diagram shows activities in context of process flow.

**Usage of Method**
Second phase also called as “Usage of Method” describes regular steps performed on daily bases. We can say that this phase is for end-users (estimators). The goal is to train configured classifier using appropriate amount of historical data and provide a new requirement to this classifier. Neural Network (classifier) help to estimator make more accurate estimations.

**Evaluation and Improvements**
Finally there is the last but very important phase is called “Evaluation and Improvements”. The goal of this phase is to measure accuracy of estimations and analyse difference between estimated and actual working-time. Results of this analysis can help improve accuracy of estimations. In other words, the number of input, output parameters or possible values of those parameters can be changed.

For summary, there are three main phases that are interconnected and visually described in Figure 18. Blue arrow shows relations between these phases in the figure. The first phase deployment and initial configuration is necessary step before the methodology is using. Usage of the method is not the last step, because the method should be continuously improved. This phase is called evaluation and improvements. Accuracy of estimations is analysed and inputs or outputs can be re-defined.

**Description of Process – Steps of Process bases on the Methodology**
Activity diagram bellow described the core process. Activities are better described in following paragraphs. Following activity diagram describes the process of deployment of methodology and its usage.
Identification of Existing Requirements

Deployment of methodology starts with exploration of requirements and identification of key features of these requirements. Proposed parameters for future processing are identified by consultations with project managers, experts, and also by performed experiments.
Descriptions of Developers

Developer can be described by productivity, maturity level or technical background. Key parameter used in this proposed estimation method is called productivity and it's defined by coefficient of productivity. For example, this coefficient of productivity can has a value 0.8 for senior developer, or 1.5 for junior developer. It means that senior developer is able to develop functionality estimated for 10 hours in advance, in 8 hours. On the other hand, junior developer is able to implement the same functionality in 15 hours. Lower value of productivity means more skilled developer. Results of exploratory analysis of data from experienced software company say that tasks with higher priority are usually assigned to senior developers.

Loading of Historical Data

Historical database of project management tool in the software company usually includes thousands of rows and hundreds of parameters. There is need to load important entities and parameters for support of estimations using machine learning algorithms. Before the training of some machine-learning algorithm starts, parameterization and pre-processing of data is required. Historical database should include information about software developers, projects, products and also about actual time of implementation of particular tasks. Those information are usually saved as worksheets.

Parameterization and Pre-Processing of Data

This step is the first step of technical part of estimation. The goal of parameterization is to obtain desired parameters mentioned in the next paragraph. It is possible to train machine-learning algorithms using vectors with these parameters. Flowing parameters should be transformed to number-values and these values should be normalized.

Proposed parameters

- Name of requirement – short title with information about the task.
- Description of requirement – extend information about the task.
- Priority – the number that shows importance of the task.
- Sum of estimated time – all estimated time.
- Estimated time for analysis
- Estimated time for implementation
- Estimated time for test
- Productivity of assigned developer.
- Type of requirement – new functionality, change, fix a bug.
- Homeworking (optional parameter) – yes or no.

Numerical parameters can be divided into quantitative and categorical. Classifiers, which is applied in this methodology is Feed-forward neural network. This neural network requires quantitative or binary input parameters. Categorical parameter – type of requirement, is transformed to binary variable during performing transformation process. [72] Binary vectors are showed in the following Table 14. We have 3 values in binary vectors for each value of categorical parameter. For example, category ,,Change of existing functionality“ includes vector with 3 binary values – \{0,1,0\}. This transformation is important step of pre-processing.

<table>
<thead>
<tr>
<th>Type of requirement</th>
<th>Category ID</th>
<th>Binary 1</th>
<th>Binary 2</th>
<th>Binary 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>New functionality</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Change of existing functionality</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Fix a bug</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 14: Example of transformation – type of requirement.

Training of Neural Network
Feed forward Backpropagation neural network for this classification purpose has been selected by using gathered knowledge from experiments. This kind of neural network consists of neurons on three layers – input, hidden, and output layer. Training samples are provided to this neural network during the training process. Necessary step is to validate the output by known desired values. Activation functions hyperbolic tangent and softmax have been selected. Hyperbolic tangent is appropriate for hidden layer and softmax is appropriate for the output layer. Mean squared error (MSE) function for calculation of error value has been selected. Number of inputs and outputs is usually between 7-10 according to suggested parameters.

Estimation of Working Time
Person in role of estimator (project manager or systems analyst) is responsible for time estimation of working time on software requirements. Estimator is ready to estimate when neural network is trained using historical data. New requirement is parameterized and transformed to desired form – vector of quantitative numerical values. Now, it is time to provide the transformed vector to neural network. Neural Network classifies it into time-group. Important part of estimation is also evaluation and improvement of the methodology.
Evaluation and Improvements

This phase from Figure 18 includes measurement of accuracy. This measurement is performed using estimated time and actual time. The goal is to reach no difference between estimated and actual time. In other words, the goal is accuracy 100% that is unrealistic value in field of effort estimation but realistic is to reach value of accuracy as high as possible. Evaluation of accuracy should be done periodically. Improvement of the method is necessary step Inputs can be added or removed according to needs.

5.3. Guideline and Best Practices

Previous paragraphs include information about inputs, outputs, configurations, and generally about usage of proposed methodology. This chapter provides guideline with all steps required for deployment and usage of the methodology by people in software company.

Existing software requirements and information about developers are analysed.

Example of this data analysis is showed in the previous chapter „4. Exploratory Analysis of Software Requirements“. Important issue is to identify parameters with influence on working time of developer on particular task. Next important step is to divide parameters into groups:

- Text parameters (e.g. name, description)
- Quantitative numerical parameters (e.g. estimated hours)
- Categorical numerical parameters (e.g. type, status)

After a thorough data analysis we can continue with data pre-processing.

Data are prepared for future processing.

Important parameters are identified. Next issue is to divide these parameters into groups input and output parameters. The methodology proposes solution with output parameter called „working time“. Actually this output parameter is categorical. Input parameters are divided into text, quantitative numerical, categorical numerical. The goal is to keep all parameters numerical quantitative or binary. Example of transformation from quantitative to binary values is showed in Table 14.

Neural Network Classifier is created, configured and trained.

Previous step related to processing data is necessary for future application of neural network. Feed-forward backpropagation neural network has been selected for classification of requirements. Difficult task is to find optimal configuration of NN and also appropriate activation functions. Number of neurons of input layer depends on number of input
parameters. Number of neurons of output layer depends on number of categories. Number of categories is set up by number of time-groups in case of time-estimations. Neurons on hidden layer use hyperbolic tangent activation function, and neurons on output layer use softmax activation function.

**Effort as estimated with support of Neural Network by following to best practices.**

Estimator estimates an effort by converting of software requirement into parameterized vector, makes estimation and provides it to neural network. Vector is classified into group. It allows to estimator make estimations more accurate. Final decision is on estimator, but neural network can help to do estimation more accurate.

**Accuracy of estimations is measured and estimation process is optimized.**

It is important to improve the software process and especially the way for estimations. The goal is to have estimation accurate as much as possible. Results of estimations from previous iteration should be evaluated. Neural network can be reconfigured or some parameters can be added or removed based on results of evaluation.

![Database of the project management tool](image1)

![Results of classification using Neural Network](image2)

<table>
<thead>
<tr>
<th>Requirement example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter name</td>
</tr>
<tr>
<td>Code</td>
</tr>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Project code</td>
</tr>
<tr>
<td>Product code</td>
</tr>
<tr>
<td>Category code</td>
</tr>
<tr>
<td>Employee ratio</td>
</tr>
<tr>
<td>Employee id</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Estimated hours SUM</td>
</tr>
<tr>
<td>Actual hours SUM</td>
</tr>
</tbody>
</table>

![Example of training item for Neural Network](image3)

<table>
<thead>
<tr>
<th>Example of training item for Neural Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project id</td>
</tr>
<tr>
<td>Product id</td>
</tr>
<tr>
<td>Category id</td>
</tr>
<tr>
<td>Employee ratio</td>
</tr>
<tr>
<td>Employee id</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>NS* output</td>
</tr>
<tr>
<td>Estimated hours SUM</td>
</tr>
<tr>
<td>Actual hours SUM</td>
</tr>
</tbody>
</table>

Figure 20: Example of Effort Estimation.
Best Practices

- Estimations should be kept as short as possible.
- Additional time reserve 30% for estimations is required.
- Critical tasks should be assigned to developers with lower value of ratio. (Senior)
- One requirement (task) is assigned to one developer
- Component-based architecture.

Following groups are suggested based on time estimation:

- 1 - 2 hours: This group includes requirements estimated for more than 1 hour and less than 2 hours of working-time. This kind of requirements usually type of requirements like small change, in source code, database, or update installation. Requirements included in this group can be type of support.
- 3 – 4 hours: This group includes requirements estimated between 3 and 4 hours.
- 5 – 8 hours: This group includes requirements estimated between 5 and 8 hours.
- 9 – 16 hours: This group includes requirements estimated between 9 and 16 hours.
- 17 – 48 hours: This group includes requirements estimated between 17 and 48 hours.
- 49 – 96 hours: This group includes requirements estimated between 49 and 96 hours.
- 97 - 160 hours: This group includes requirements estimated between 97 and 160 hours.

5.4. Experiment - Evaluation of Proposed Method

Whole evaluation consists of three steps. The first step is selection and collection of historical data from SQL database. This step is better described in the data model sub-section - includes description of data model and description of data pre-processing techniques before classification process starts.

Second phase, better described in classifiers sub-section. It describes design of classifiers, particularly Feed-forward Neural Network (NN) and Naive Bayes Classifier (NB). Sub-section also includes was of use NN and NB classifiers.

Third phase, described in training sub-section describes two-phases training approach. The last sub-section, named estimation - includes interaction of project manager with effort estimation supportive tool based on proposed method.

Classifiers should to be designed for concrete dataset. In this case we need to design classifiers for classification of requirements into few categories. First of all, we need to classify text items into two categories. Appropriate classification algorithm for this purpose is Naive Bayes classifier.
Naïve Bayes is often used in spam filtering. Spam filtering is not the only application of Naïve Bayesian. In fact, Naïve Bayes Text Classifier has been around many years. It allows automatically classify documents of all kinds. In this case, we need to classify text documents into groups „right estimated“ and „underestimated“. After items were classified, text values of parameters are replaced by integer values. A set of vectors with integer values is available as a result of the first classification phase. These vectors are input-training items for neural network classifier.

The neural network classifies items into several groups. The neural network is based on feed-forward architecture using back-propagation training algorithm. Number of input neurons will be selected according to number of input parameters. Number of output neurons will be selected according to number of time-groups (output classes).

Number of groups is based on key reason – practical experience of project managers and analysts. Sum of estimated hours includes time for consultations, analysis, write documentation, project management, development, support, test, deploy database and application, and also time for customers training. [60]

**Experimental Data Model**

Provided data model (Figure 21) is extracted from the real database of project management application. This application stores information about software projects, developers, software requirements, and tasks. The data model includes following entities:

![Class diagram – Data model](image)

Figure 21: Class diagram – Data model
**Employee** is entity that represents employee of company. This employee can be in different roles, e.g. analyst, developer, technical support or consultant. Each employee is characterized by code, name and productivity. Productivity is double value parameter and it means also seniority of current developer. (e.g. junior developer has productivity 1.5 and senior developer has productivity 0.85).

**Product** entity is characterized by code, name and description. Products are concrete software ERP and CRM applications developed by company. Each requirement is assigned to the product.

**Project** entity represents a software project. It is characterized by name and code. The development of a project goes through a number of requirements, thus it consists of requirements. Each requirement is assigned to the project.

**Requirement** represents a single requirements specification of the system. It has parameters like code, name, description, category, type, status and each requirement can be assigned to project, product and employee.

**Task** entity has code, name, description, and other parameters like category, estimated hours, real hours, and priority value and. Each task is an assigned employee and requirement. In other words requirements consists of number of tasks. Some tasks can be marked as children. It means that they are assigned to parent task, through parent code.

**Worksheet** is important entity for summarization actual hours. Using information from worksheet, we are able to compare estimated and actual hours. Each worksheet has to be assigned to task and employee.

The most important entity within the whole data model (Figure 21) is “Requirement”. This entity includes parameters name, description, priority, type, estimated hours, and also estimations for particular stages of software development like analysis, implementation and testing. Next parameters are derived from other entities, e.g. productivity of developer or actual hours. The result of analysis of provided data model is set of quantitate and binary variables. Some of them have been selected directly from requirement entity, another have been derived from another entities, or using transformation techniques for data pre-processing. Let’s look at a toy example of set of quantitative and binary parameters with example values. [60]
Parameters of example data set:

- Name length: 56 chars - the length of string “name”
- Description length: 232 chars – the length of string “description”
- Priority: 1.0 – priority of particular requirements varies between 0 and 1
- Diff sum: 4 hours - difference of estimated and actual hours
- Sum actual: 20 hours - sum of actual hours derived from worksheet of employees
- Sum estimated: 16 hours - sum of estimated hours taking account estimated hours for analysis, implementation and testing
- Estimated analysis: 4 hours - estimated hours for requirements analysis
- Estimated test: 2 hours - estimated hours for testing of implemented functionality
- Estimated implementation: 10 hours – estimated hours for implementation
- Productivity: 1.2 – productivity of developer (described in subsection 3.1)
- Requirement type: New - (New functionality / Fix the Bug / Update currently implemented functionality)

4.5.5 Classification of Software Requirements

Dataset Description

A historical database contains 1549 items (software requirements) represented by quantitative and binary parameters. The goal is to classify items into 6 categories (time-groups). These time-groups have been created base on actual hours - that is parameter of software requirement.

Requirements are divided into time-groups: half day (0 – 4 hours), one day (more than 4 - 8 hours), two days (more than 8 – 16 hours), one week (more than 16 – 40 hours), two weeks (more than 40 – 80 hours), and (other) more than 80 hours. Whole dataset consists of 1549 items. This dataset has been divided into three subsets. The first is for training includes 929 items. It is used for computing the gradient and updating the network weights and biases. Second subset includes 465 items and it is for validation – finding of minimal validation error. Last set includes 155 test items. Test subset is used for measurement of neural network accuracy and it is not used in training phase. [60]

Training of Neural Network for Classification purpose

The two-layered feed-forward network with sigmoid hidden and soft-max output neurons is trained using scaled conjugate gradient backpropagation. Training of neural network automatically stops when generalization stops improving, as indicated by an increase in the
cross-entropy error of the validation items. Following Figure 22 shows that final value of gradient is 0.029121 at epoch 146.

![Gradient plot](image)

**Figure 22: Training State Plot – Gradient**

The max value of “max fail” parameter in MatLab environment was kept as default value = 6. Training state plot (Figure 23) shows that training stopped after six fails, because the neural network stop improve.

![Validation Checks plot](image)

**Figure 23: Training State Plot – Validation Checks**

Plot with validation performance (Figure 24) shows results of validation during training process. The best results of validation are indicated at epoch 140. Best validation performance at this epoch is 0.043996. Training of the neural network had continued for next six iterations and then stopped.
4.2.1 Results of Classification

In this sub-section, we discuss how to effectively apply and improve effort estimation supported by machine-learning methods. After the estimated work is completed, the estimates should be evaluated against actual outcomes of the work. Employees hold the responsibility for performing the estimated work, and they should review outcomes of estimations. The goal of a review is to ensure that estimation methods, techniques, data, and guidelines have been properly used. Classification results are visualized using confusion matrixes (Figure 25). Classification of software requirements was performed using feedforward neural network and backpropagation training algorithm. Six diagonal cells show number of correct classifications and also percentage bellow. [60]

The first is training confusion matrix shows results of classification during the training. Number of correct classifications is showed in diagonal of matrix. Classification of samples into particular classes is 58.8% for 1. class, 12.3% for 2. class, 9.9% for 3. class, 9.7% for 4. class 1.0% for 5. class, and 0% for 6. class. Overall training accuracy is 91.9% (right, bottom corner of test confusion matrix).

Second, validation confusion matrix shows the same results, but for validation checks – using dataset-items for validation. Classification of samples into particular classes is 61.1% for 1. class, 12.3% for 2. class, 7.1% for 3. class, 11.0% for 4. class 1.5% for 5. class, and 0% for 6. class. Overall accuracy of validation checks is 92.9%.
Third, test confusion matrix is the most important, because it shows most relevant results of classification. Classification of samples into classes is 58.1% for 1. class, 12.3% for 2. class, 11.0% for 3. class, 9.7% for 4. class 1.3% for 5. class, and 0% for 6. class. Overall accuracy of classification measured using test dataset is 92.3%. [60] Lets look at this matrix in detail. The first class is classified with the highest accuracy (95.8%). On the other hand, measurement of accuracy of the last class is marked as NaN (Not a Number). There was not enough data for measurement of that class (time-group of tasks estimated for more than 80 man-hours or more than 2 man-weeks). The results confirm that there is no point to estimate tasks for developers for more than 2 weeks or in other unit for more than 80 man-hours. [60]
Provided confusion matrixes show that accuracy of classification into particular classes is between 66.7 and 96.8%. Class with highest precision is the first class, which includes tasks estimated between 1 and 2 hours. Second most frequent class includes tasks estimated between 8 and 16 hours and third class includes tasks estimate between 17 and 40 hours. Class that includes tasks estimated for more than 160 hours is no more important for future effort estimation. Result of performed experiment also helped to create best practices.
6. Summary and Conclusion

Nowadays, software companies need to do estimations more and more accurately. A lot of various methods for effort estimation are available. All of them are appropriate for different environment and type of a software company.

The main idea of this dissertation thesis was to develop advanced method for support of effort estimation process in a software development company. Developed method should help to project managers to estimate an effort more accurately. This dissertation thesis also presents and discusses usage of proposed methodology based on feed-forward neural network for support effort estimation. This methodology uses historical data from a software company to train configured neural network and then the neural network is used to classify software requirements into time-groups. Classification as a supportive technique helps to project managers identifies tasks with higher probability of underestimation. Finally, this proposed approach is evaluated by experiment performed on data from real software company. Results of our previous experiments in field of classification of use cases are provided. This thesis is divided into four parts.

The first part of this thesis that is called “State of the Art” (Section 2) describes current progress of research in field of effort estimation. The section also introduces several different approaches for effort estimation and it provide comparison of those approaches. Widely used approaches for effort estimation are divided into three groups: The first groups includes algorithmic (formal) models like COCOMO, COCOMO II, UP, UCP. Those formal approaches are appropriate for usage in institutions like NASA or commercial banks. An integration of those methods into daily-life of a software company is very difficult and with uncertain result. Formal methodologies are not so flexible as companies expect and those methods also include a lot of rules and formulas. Following of rules is necessary for accurate results. Next group is called expert judgement and soft computing models. Actually, those two groups are closely related and combined in some parts. Expert judgement is depended on highly skilled person with knowledge of certain domain. This person also can make some mistakes. Soft computing models support decisions of experts and also can help to experts eliminate number of inaccurate estimations. Section 2.4 provides information about accuracy of models and also comparison of those models. Average accuracy of predication of work-time is about 90%.

Second part includes sections “Classification as a Supportive Technique” (Section 3) and “Exploratory Analysis of Software Requirements” (Section 4). Section 3 provides overview of various classification techniques and shows results of experiment with Naïve Bayes Classifier and Feed-Forward Neural Network. Is also says that appropriate neural network for classification of software requirements in form of use cases is Feed-Forward Neural Network with backpropagation training algorithm. That section also includes finding
of optimal configuration for selected neural network. Section 4 provides results of exploration of software requirements. Particular parameters are described also graphically by box-plots or component plane. Exploratory analysis has been performed using statistical method – Principal Component Analysis and also using Neural Network – Kohonen’s Self-Organizing Map. Results have been visualized by component plane. Component plane has been also used for identification of related parameters after the clustering process. The section provides review of literature in field of exploratory analysis and describes principles for exploration of available data sets. The main objective of the Section 4 was to provide results of exploration of software requirements. Obtained results have been used for proposing of the methodology. Proposed methodology is more detailed described in following third part of this dissertation.

The third part “Proposed Method for Effort Estimation” (Section 5) is focused on proposal and description of methodology that should help to project managers in software companies. Methodology uses real data from a database of the software company. The database includes information about tasks for developers, software projects, developers and worksheets. Section 5.1 describes pre-condition for deployment of this methodology. It describes roles and artefacts that are necessary for successful application of the methodology. It should be possible to deploy the methodology into environment of software company and use the methodology effectively based on information provided in this section 5.2. People in a software company that uses the methodology for effort estimation should be able to integrate this methodology into their daily activities related to software development. Accuracy of proposed method was measured with result 92.3%. This result shows that accuracy is very similar like existing methods described in section 2. Difference between available methods is in type and size of company-environment. Pros and cons of approaches are better described in the next paragraph.

Finally, proposed methodology is evaluated by experiment based on data from the real software company and in order to this, it is showed that proposed method is appropriate and valuable as a supportive method for software process.

Accuracy, Pros and Cons of Proposed Approach

Many different methods for effort estimation are available. Each of them is appropriate for different environment or different size of a software company. Formal methods like UCP or COCOMO I/II are more appropriate for usage in bigger software companies or institutions like NASA. These formal methods are usually applied in corporations following their internal directives. Disadvantage of those formal methods is disability to accept changes quickly. Those methods are more appropriate for support of software process in smaller or middle-size software companies. Those disadvantages call for more flexible methodology that is able to accept changes in a changing environment. Flexible and innovative approach for small or
middle-size software companies is provided in this thesis. Proposed methodology combines usage of historical data, expert knowledge and machine-learning techniques for classification of software requirements. This combination of methods, techniques and approaches creates a powerful tool for elimination of inaccuracies in estimations of tasks. The disadvantage of proposed methodology is the lack of formal approaches. Mentioned formal approaches are more appropriate for big companies or institutions.

**Achieved Goals**
The goal of this thesis is to provide insight into effort estimation methodologies, analyse data from a software company, perform experiments and propose methodology for software effort estimations. The main goal of this thesis has been achieved. The methodology was proposed and evaluated by experiments. The method is ready for usage in a software company.

**Particular Goals of the thesis:**

- Comparison of widely used approaches for effort estimation
- Selection of an appropriate technique for classification of software requirements.
- Exploratory analysis of real data – software requirements.
- Design of the methodology for effort estimation.
- Description of the methodology and guideline for a software company.
- Evaluation of the methodology by experiment.

**Future Work**
In order to future plans related to usage of the method in software companies, there is an intention to improve proposed method using new, different data-sets and information about development teams from other software companies. It is possible to improve existing features and propose new features based on results of experiments based on different data.
Author’s Indexed Publications

Related to thesis, cited and listed in references


Other publications


**Author’s Participation on Projects**

The research presented in this thesis has been supported by Grants of VŠB – Technical University of Ostrava. Participated projects:

- **Project SP2016/100** Knowledge modelling and its applications in software engineering II, VŠB - Technical University of Ostrava, Czech Republic, 2016.
- **Project SP2014/157** Knowledge modelling, simulations and design of software processes, VŠB - Technical University of Ostrava, Czech Republic, 2014.
- **Project SP2013/207** An utilization of artificial intelligence in knowledge mining from processes and process modelling and mining, VŠB - Technical University of Ostrava, Czech Republic, 2013.
References


78


