Neural network based model for productivity in software development

Author: Steffen Janssen

Studentnumber: 0710318

Supervisors:
Ir. E. Hack
Prof. dr. T.M. Heskes
Dr. H.C. de Coninck

Belastingdienst
Radboud University
Radboud University

Master’s thesis
Computing Science - Management & Technology

Thesis number: 672
09-10-2013
Abstract

This paper looks into productivity of 634 software projects completed between 2009 and 2013 within the IT department of the Dutch tax administration. A predictive neural network based model is developed in order to produce better estimates for future development efforts. Direct cost drivers and numerous organizational changes are identified in order to analyze their impact on productivity. The neural network is capable of generating better estimates than other software cost estimation methods, such as COCOMO II and QSM-SLIM. The research concludes that there are opportunities to reduce the overall costs of software development by improving planning.
First and foremost I would like to thank my three supervisors for guiding and helping me along the way in writing this thesis. Discussing my progress, problems, and ideas with Elco Hack a couple of times every week helped me tremendously in understanding the rationale behind the assignment. It made me better realize the business need for this analysis. Heleen de Coninck helped me by looking outside of merely all the cost driver information, and considering the organization from a more holistic perspective than just from the productivity point of view. Tom Heskes patiently explained to me a great number of technical details with regards to neural networks, for example how to deal with such an unstable learning method and to provide insight into the reliability of those results.

Without their time and help none of this research would have been possible, and I’m grateful to them and the Belastingdienst for giving me the opportunity to work on this interesting subject.

I would like to thank Ad Voorsluijs, and everyone in the Productiviteit function point analysis team: Anne, Bert, Gerard, Guido, Hans, Jaco, Linda, Marga, Peter, Peter, Remco, Sarkis, Stef and Wouter.
# Contents

Abstract i

Acknowledgements ii

1 Introduction 1
   1.1 Problem overview ........................................ 1
   1.2 Belastingdienst organization .......................... 2

2 Research Goal 3
   2.1 Research questions ................................. 3

3 Theory 4
   3.1 Software cost estimation ............................. 4
      3.1.1 Challenges ..................................... 5
      3.1.2 Cost drivers ................................... 6
   3.2 Scientific management ............................... 7
      3.2.1 Lean ............................................ 8
      3.2.2 Function point analysis ......................... 9
   3.3 Cost estimation methods ......................... 10
      3.3.1 COCOMO II ...................................... 10
      3.3.2 QSM SLIM ....................................... 11
      3.3.3 Non-algorithmic cost estimation ............... 12
      3.3.4 Algorithmic cost estimation .................. 12
      3.3.5 Pattern recognition ............................ 13
   3.4 Multilayer perceptron ............................ 14
      3.4.1 Perceptron ...................................... 15
      3.4.2 Neural network .................................. 16
      3.4.3 Learning ......................................... 17
      3.4.4 Confidence measure .............................. 18

4 Problem Domain 22
   4.1 Project information ................................. 23
      4.1.1 Productivity ...................................... 24
   4.2 Staff expenses ..................................... 26
   4.3 System information ................................. 27
   4.4 Cost drivers ....................................... 28
   4.5 Usability ............................................ 28
   4.6 Approach ............................................. 30
## Contents

5 **Productivity in B/CAO** 32  
5.1 Environment 32  
5.2 Organizational changes 32  
5.2.1 Conclusion 37  

6 **Data Preprocessing** 39  
6.1 Enriching and merging 39  
6.2 Feature selection 41  
6.3 Feature construction 42  
6.4 Pruning projects 42  
6.5 Normalization 43  

7 **Proposed Model** 44  
7.1 Motivation 44  
7.2 Implementation 44  
7.2.1 Interface 45  
7.3 Instructions 46  
7.4 Maintainability 47  
7.5 Analysis 48  
7.5.1 Method 48  
7.5.2 Functional size 50  
7.5.3 Duration 52  
7.5.4 Average hourly rate 54  
7.5.5 % External employees 56  
7.5.6 % New functionality 58  
7.5.7 Multiple programming languages 60  
7.6 Performance 61  

8 **Discussion** 64  
8.1 Comparison with QSM SLIM 64  
8.2 Opportunities 65  
8.3 Limitations 66  

9 **Conclusion** 68  
9.1 Recommendations 68  
9.1.1 Data 69  
9.1.2 Model 69  
9.1.3 Transformation 70  
9.2 Future Research 70  

A **Encog model** 75  
B **Cost estimation model** 77
1. Introduction

Software cost estimation is not a new issue in IT. Ever since the complexity and size of computer programs has been growing there has been an increasing need for reliable cost estimation. Whereas the very first computer programs were extremely limited in size and scope, current day software projects can entail millions of lines of code, up to a thousand simultaneous developers, and massive supporting services, infrastructure and management teams. Such projects are costly, thorough planning is required to guide the available resources in an optimal way. Because the financial consequences of inaccurate cost predictions are considerable, there is sufficient incentive to spend some effort in doing formal cost predictions.

IT has not had a good reputation when it comes to delivering products and services in a timely and costly fashion. Large software projects have a tendency to run over their scheduled costs, take too much time and deliver disappointing quality.

These risks justify doing research into new ways to make accurate and reliable cost predictions.

1.1 Problem overview

The Dutch tax administration (Belastingdienst) is involved in numerous software projects. Its IT department (Belastingdienst/Centrum voor Applicatieontwikkeling en Onderhoud) has a need for a more reliable way to estimate the costs for new software projects. The current tools available which have been used in the past do not produce satisfactory predictions. Previous research into various other software cost estimation tools revealed that these methods were not entirely suitable for the Belastindienst B/CAO organization [1].

The other tools that have been investigated are:

- COCOMO II
- QSM SLIM
The industry standards for cost modeling in IT are designed based on a large number of older projects which have been implemented in completely different companies in entirely different environments. For these reasons the Belastingdienst is looking for an organization-specific way to reliably estimate the cost of future software projects.

Ultimately the goal of this research is to gain insights into effects on productivity in software development.

1.2 Belastingdienst organization

The Belastingdienst is part of the Dutch Ministry of Finance. It is responsible for taxation, detecting financial fraud, government benefits, and customs. Belastingdienst / Centrum voor Applicatieontwikkeling en Onderhoud is its supporting IT department. A wide range of information systems support and help processing all kinds of tax applications. Some IT systems are available for citizens to help them file their tax return, others to apply for government benefits. But also software used to track goods and shipments brought into ports is developed by B/CAO. B/CAO develops and maintains Belastingdienst IT services and products. Its IT infrastructure services are provided and maintained by Belastingdienst / Centrum voor Infrastructuur en Exploitatie (B/CIE).
2. Research Goal

The research goal of this thesis is threefold:

1. To investigate and analyze the available historical data on completed software projects in order to determine if it can be used to build a predictive cost model.

2. To design, implement and analyze a new software cost estimation model that can reliably predict the effort required to complete future IT projects for B/CAO.

3. To gain insights into cost drivers on productivity in software development for B/CAO.

2.1 Research questions

The following questions guide this research.

q1. Does the dataset offer enough data that it can be used to build an accurate cost model for IT projects in the B/CAO organization?

q2. What is a suitable technique for building a software cost estimation tool specific for the B/CAO organization?

q3. What business processes and organizational changes have influenced the increase in productivity since 2009, and how have they impacted the entire organization?

q4. Does the new software cost estimation tool produce better results predicting the costs of software projects in the B/CAO organization than other tools and QSM-SLIM in particular?

q5. How can B/CAO optimize productivity in software development?
3. Theory

3.1 Software cost estimation

Software cost estimation is not a new issue in IT. Ever since the complexity and size of computer programs has been growing there has been an increasing need for reliable cost estimation. Whereas the very first computer programs were extremely limited in size and scope, current day software projects can entail millions of lines of code, up to a thousand simultaneous developers, and massive supporting services, infrastructure and management teams. Such projects are costly, thorough planning is required to guide the available resources in an optimal way. Because the financial consequences of inaccurate cost predictions are considerable, there is sufficient incentive to spend effort in doing formal cost predictions.

As well as running over budget, there is the risk that projects are not completed in time. Certain time critical software applications have a fixed deadline and must be delivered before the deadline has passed. Furthermore, running late on software projects may lead to entire product lines being delayed or scrapped altogether. Business opportunities are overwhelmingly time critical; not delivering the right product in time may lead to a competitor taking a commanding market share, effectively eliminating your company from this particular market segment. A large software company may typically produce hundreds of applications of varying sizes. These risks have led to software cost estimation becoming more common in any business that produces large volumes of software.

Despite this need there has not been a single convincing theory for cost estimation [2]. Most methodologies are model based and depend on the number of lines of code. Very few cost estimation methodologies have considered looking at function points as a measure of size. Today, there is no one model that can accurately predict the costs involved in any software development effort. Three reasons why this problem will remain relevant are:

- There are a large number of interrelated factors that influence software development
- We are not aware of all the factors that could influence software development
- Development environments are continuously evolving
Besides the need for accurate predictions in the day-to-day business of building software, there is also the aspect of possible exposure to litigation. In this increasingly connected modern world, software is usually linked to other systems. Not delivering products can have significant consequences for other parties and may open up various legal options where cost estimations play a prominent role [3].

Software estimating is simple in concept, but difficult to do properly in reality. The scope and size of projects influences the number of factors that must be evaluated to get an accurate prediction of the eventual costs. The complexity of manually doing estimates for larger software projects usually exceeds the capabilities of most project managers. The amount of work required for activities other than coding are often misjudged and can vary greatly between different types of projects. Although automated cost estimation estimates are imperfect, they are mostly more accurate than human predictions and are far less costly. One of the greatest advantages for doing automated software cost estimation is that it is less susceptible to subjective reasoning. The current best practice in software engineering for cost estimation is to use a combination of an automated system and expert opinions [4]. Although another research which reviewed a number of software cost estimation models concluded that expert estimation was generally superior to automated models [5]. A possible reason why expert opinion could outperform formal models is because an expert has a grasp of all the possible cost drivers that may be applicable to a project, while the model is limited to predefined parameters. For example an expert may be aware of the intrinsic complexity of collaborating between two project groups who are situated in different locations while a model may not take such factors into account. But automated models do not suffer from “wishful thinking” and will deliver truly objective and reproducible estimates. In situations where there are strong human biases, e.g. human estimators have a strong personal interest in the outcome, automated models will usually perform better.

### 3.1.1 Challenges

Cost estimation is an activity in many types of industry. But producing software is very different from producing tangible products. Engineering software is different because it can depend on creating and applying new solutions to new problems that have not been solved before. Furthermore, software is custom-made whereas most industries are about mass production of goods. So each new software product has been specifically designed and built according to a certain specification. The fact that each software project requires innovative thought and problem solving skills makes its development cycle far more unpredictable than other manufacturing processes. This introduces immense challenges in trying to predict how much a software project is going to cost, especially when it is a complex and lengthy problem.
A significant issue is the limited availability of data for training a cost model. IT projects are costly and take a relatively long amount of time, so organizations cannot produce or acquire more data very easily. Buying or using information from other organizations is of very little use, or even counterproductive. Due to different ways of data collection, other ordinal scales used and different environments, the data on software projects cannot be accurately compared between companies. And because of the rapid advances in computing and software engineering principles and processes, data is outdated very quickly. This is further compounded by organizational changes that take place. Examples of events that can influence the productivity are: employees leaving, employees receiving different positions, changing work processes and outsourcing of work. All these possibilities can change the dynamics, productivity and skill level of a development team, rendering past performance unrepresentative of the current situation and leading to inaccurate cost predictions. Limited amounts of data in a highly dimensional feature space means that the data density is very low. This sparsity introduces difficulties in precisely predicting a target variable (costs or hours) for new projects. Because the topic of cost estimation is inherently difficult and pinpoint accuracy is not required we can accept a certain margin of error.

3.1.2 Cost drivers

Generally, cost estimation is based on several different factors that influence the project in different ways. The types of factors that define the scope, difficulty and eventually the overall costs, can be segregated into four types:

- Personnel
- Technology
- Environment
- Process

These aspects together can define the eventual software quality and the effort required to build a particular piece of software. Although this research is primarily focused on the cost aspect, it is evident that there is a strong correlation between software quality and costs. Most cost drivers considered in this paper belong to the categories Personnel and Technology.

A general function to compute and predict the effort required to build an IT system, where \(x_0, x_1, x_2, \ldots, x_n\) are the cost drivers, is [3]

\[
\text{Effort} = f(x_0, x_1, x_2, \ldots, x_n).
\] (3.1)
3.2 Scientific management

In order to estimate costs you first need data. The idea of using hard data to make important management decisions is based on the idea of scientific management. Frederick Taylor was lead developer of the theories on scientific management [6]. It started during the 1880s and 1890s and focused on manufacturing industries. During the industrial revolution the need for higher production became apparent, and Taylor developed his theory by working on consulting jobs for clients who were looking to increase production. Scientific management is a management methodology in which performance and efficiency of individuals and teams can be measured through productivity metrics. It was particularly suitable in a time where labor shifted from craftsmanship to mass production. And in our current day management methodologies it is still highly relevant because it introduced (to management theory) the concepts of analysis, synthesis, and efficiency. The theory analyzes how workers should be motivated in order to maximize productivity, efficiency and eventually profit.

The main goal of Taylorism (scientific management) is to increase efficiency by dividing up complex tasks in smaller units of work which can be completed by a single individual. Taylor’s main goal was to give employees the tools they needed in order to eliminate waste in their direct work process. Taylorism is based on the assumptions that workers are inherently lazy, uneducated, and are only motivated by money. Clearly such views are unpopular among workers, and the scientific management methodology has drawn its share of criticism. It is said to take away control from the employees over their pace of work, workplace, and tools; and most importantly that they are viewed as mere machines rather than human beings. By focusing on a single task workers have little sense of the finished product and derive less satisfaction from their employment. Management gains even more control over the process, which could potentially lead to repressive workplace situations.

Its advantages are the capabilities it brings to management by being able to analyze bottlenecks, problems, inefficiencies, and waste. Employees are treated more fairly, it acknowledges high performers leading to possible growth opportunities, and is meant to lead to higher wages. In actuality scientific management by Taylor led to a great number of conflicts and strikes, and possibly even strengthened labor unions [7].

Over time it went through numerous iterations and developments, and now Taylorism has found its place in current day management ideas such as Lean and Six Sigma [8]. They also focus on efficiency, productivity and elimination of waste. But also include core values such as respect for employees and personal fulfillment. A key aspect in the Lean method as applied in B/CAO is continuous measurement and improvement of certain performance indicators. In software development a key performance indicator is the amount of software
functionality produced per hour. Function point analysis enables us to size software and measure productivity in certain types of IT jobs.

### 3.2.1 Lean

A study by McKinsey & Company into the B/CAO work processes led to the introduction of Lean [9] in 2010. Lean is a process management methodology and is currently being used throughout the entire organization. Lean in IT can be applied to both Agile and waterfall development. It finds its origins in the late 1940s in Toyota when it was called the Toyota Production System. It was termed Lean only later in 1990 [8]. Researchers involved in the International Motor Vehicle Program found that Toyota had a higher productivity than its competitors. Although at the time worldwide production of automobiles was dominated by the large American car manufacturers, it became apparent that the Japanese were rapidly catching up through higher efficiency and productivity. The researchers studied the principles of this “lean organization”, leading to the Lean management theory that we know today [10].

The main Lean principles are about integrating **Value**, **Value Stream**, **Flow** and **Pull**. Together these principles should come together in the final principle of **Perfection** [8, 11].

- **Value** is the value to the customer. Only those things which are required and needed and approved by the customer qualify as value. According to Lean an organization should be made to provide a continuous stream of value-creating activities throughout the entire cycle of production processes. Providing value is the cornerstone of Lean.
- The **Value Stream** is the process a product or service goes through from initial concept to final delivery. The value stream looks at the process to identify where value is added and where there is not. Some areas where no “value” is added are not necessarily waste. For instance testing and HR management are essential yet they do not directly add value to the end product. These processes also belong to the Lean concept of value stream.
- **Flow** is the idea of reducing or eliminating processes which deliver sub-optimal value in the value stream. By mapping the entire value stream, an analyst can identify exactly where there is waste, and how to make the value stream “flow” into delivering more value. Mass production and specialization will usually optimize work processes.
- **Pull** is contrary to the common manufacturing principle known as push. Products are “pulled” out of the organization towards the market and customers. Production is optimized to the level where the organization is able to produce as the customer demands it. The opposite, push, methodology assumes that it is possible to accurately predict and forecast demand and match productivity to this demand.
the context of software engineering the pull methodology requires being able to deal with varying demand. Pull means that no upstream process may produce any goods or services, until it gets a signal to do so from downstream. In practice, it is nearly impossible in a large organization to accomplish “pull” to a state of perfection.

- **Perfection** is the ultimate goal of the Lean process management structure. As the other principles are applied and continuously improved, the organization strives towards perfection. Implementing the principles of Lean is a difficult task. While Lean strives towards perfection it realizes that this point cannot be reached. Regardless it is a framework which aims to continuously improve processes through iterations and feedback. “Kaizen” is the Lean term for continuous improvement, kaizen activities evaluate work processes and implement slight change to further optimize work.

It is not about enforcing drastic change all at once, but rather continuous evaluations and adaptations. Perfection is as important as the other Lean principles. And while it may not be achievable the attempt to get there is crucial in improving the organization.

Further background information about Lean can be found in Lean Thinking [8].

### 3.2.2 Function point analysis

A main cost driver for this research is the size of an information system: function points. Function points were defined by Allen Albrecht in 1979 at IBM [12], they identify and categorize functional user requirements into outputs, inquiries, inputs, internal files, and external interfaces. There are currently five internationally recognized ISO certified standards for functionally sizing software, among them are Nesma [13] and Cosmic [14]. These two are used in function point analysis within the Belastingdienst; the characteristics of the IT system define which method is the most suitable. An adaptation to Nesma, coined IBRA, can also be used in FPA for IT systems that do lots of complex internal computations. Function point analysis provides a consistent, documentable, repeatable measurement methodology. Nesma is nearly identical to the more international IFPUG [15].

Most studies into software cost estimation have not looked at the functional size of information systems, but rather at the physical size in terms of lines of code or number of objects. The reason for using function points as a main cost driver rather than lines of code is because this is a measure which can be accurately and objectively measured before building the system. A customer is not interested in what a 10,000,000 SLOC computer program will cost her, but rather what the required functionalities will cost.

Function points are designed to be a uniform sizing instrument for software. In practice it turns out that not every function point requires the same amount of work to build
it. But obviously FPA is all about sizing software and not sizing work. In this research we distinguish between added and modified function points. Adding a function point means adding this functionality from scratch. But an FP analyst will count the very same number of points for modifying it. Similarly full points are awarded for deleting a function point. Although in this way FPA indicates the scope of the software effort, intuitively it does not quite represent the work involved in a project effort. For that reason this research looks into the way other factors and function points together determine the workload for the purpose of doing cost estimation.

FPA is the most important means by which it is possible to apply scientific management to software development. While you can measure other aspects of work in other more traditional ways (hours, salary), size of software very different.

### 3.3 Cost estimation methods

*Boehm et al.* have identified six types of cost estimation methods [16]:

- Model based
- Expertise based
- Learning Oriented
- Dynamics based
- Regression based
- Composite

These types can be subdivided into non-algorithmic (Expertise based), algorithmic (Learning Oriented, Dynamics based, Regression based), or mixed (Model based, Composite). COCOMO II is the most commonly cited example for cost estimation in software engineering. QSM SLIM is currently being used within the B/CAO environment. For this reason these two methods are expanded upon in section 3.3.1 and 3.3.2. Both are model based.

#### 3.3.1 COCOMO II

COCOMO II [17], which is the most widely used cost estimating model for software, uses seventeen different cost drivers. Besides these cost drivers COCOMO II utilizes five additional scale factors that determine the application size, and an overall scale factor that needs to be calibrated to take into account the productivity factor of the organization or team. Most COCOMO II cost drivers are subjective assessments of personnel, tools,
complexity and other aspects of the development process. An expert provides this assessment. Fixed coefficients exist for each of the possible answers and these values are used in computing the estimate.

The essence of COCOMO II is the following formula:

\[
E = A \times \text{Size}^B \times \prod_{i=1}^{17} M_i
\]

\[
B = 1.01 + 0.01 \times \sum_{j=1}^{5} SF_j
\]

- \(E\) = Estimated effort in people months of labor
- \(M\) = Effort multiplier. These effort multipliers effect development regardless of project size
- \(A\) = Tuning parameter
- \(\text{Size}\) = Size in 1000 SLOC
- \(SF\) = Scaling factor. Scaling factors influence larger projects more than smaller projects

### 3.3.2 QSM SLIM

QSM SLIM is a toolkit that is being used within B/CAO. Its main product is the cost estimation module (SLIM Estimate). Besides the cost estimation algorithm, it also provides for software that can store project effort information (SLIM Datamanager), create estimation reports (SLIM Control), and a tool to do statistical analysis on project information (SLIM Metrics).

In 1978 Lawrence Putnam published the Software Lifecycle Management software process SLIM [18]. The essence of the QSM SLIM cost estimation formula is:

\[
E = \left( \frac{S}{P \times T^{1/3}} \right)^3 \times B
\]

- \(B\) = Scaling factor based on project size
- \(S\) = Project size in SLOC
- \(P\) = Productivity for the organization (tuning parameter)
- \(E\) = Effort in person-years
- \(T\) = Turnaround time in years

This formula is rather simplistic in nature but requires some additional effort in computing the tuning parameters. Regardless of the tuning parameters, the QSM formula implies that effort decreases as turnaround time increases, contrary to observations in B/CAO!
When accounting for the fact that smaller projects generally take less time than larger projects, we still see a clear trend that suggests that shorter projects are considerably more efficient. Only in cases where we have seen extremely high time pressure efficiency drops.

### 3.3.3 Non-algorithmic cost estimation

Non-algorithmic methods require one or more completed projects that are very similar to the new project and derive estimation through reasoning by analogy using the actual costs of the previous projects. Estimation by analogy can be done either at the top level of the project or at subsystem level. The total project level has the advantage that all cost components of the system will be considered while the subsystem level has the advantage of providing a more detailed assessment of the similarities and differences between the new project and completed projects. Obviously, doing a thorough subsystem cost analysis requires more work than a global estimate. The strength of this method is that an estimate is based on actual project experience in that very same company. However, it is not clear to what extent the previous projects are actually representative of the constraints, environment and functions to be performed by the new system. Nor can it be independently determined that the estimate is in fact based on representative older completed projects. It is a very subjective, error-prone and costly way to make predictions.

A well-known technique is the Delphi method. Delphi is about consulting experts and asking them to give their opinion and estimates based on the available information. Iterating through a number of rounds the experts can discuss each other’s estimates and make new predictions. The process ends when the experts reach consensus, there is no change in predictions or after a pre-defined number of rounds [19].

### 3.3.4 Algorithmic cost estimation

The algorithmic models vary widely in mathematical sophistication. Some are based on simple arithmetic formulas, others are based on regression models and differential equations. To be able to use algorithmic models in practice, there is a need to adjust or calibrate the model to local circumstances [20]. These models cannot be used off-the-shelf. Even with calibration the accuracy can be quite mixed between organizations and development teams.

Various commercial and non-commercial algorithmic cost estimation tools use different functions to produce and estimate the effort required to complete a software project. The form of the cost function varies but the mathematics involved is generally limited as
most tools are either linear, multiplicative or power function models. These tools are an abstraction of the reality and will usually include tuning parameters to better fit these tools to an organization [2].

Tuning parameters to fit the model to pre-existing data can only improve performance by a certain amount. This makes sense because they are only tuning parameters. It seems awkward to be restricted to using a certain model and trying to tweak it using a very limited tweaking instrument, while essentially trying to change the model to explain past performances in order to build a predictive model for new project efforts.

If the aim is to build a predictive model with the best possible predictive power, it seems more sensible to build a new model from scratch using the most promising tools and techniques. That is exactly the goal of this research. A new cost estimation tool is built using pattern recognition techniques, and a learning algorithm learns the patterns embedded within the historical data.

### 3.3.5 Pattern recognition

Pattern recognition is aimed at extracting underlying patterns in (large) sets of data. It is concerned with the theory and techniques of putting series of data into discrete or continuous categories. As the size and dimensionality of data grows it becomes increasingly harder to identify and extract underlying patterns. These patterns may depend on a complex interaction between any number of variables.

Pattern recognition methods are useful in applications such as information retrieval, data mining, image analysis and recognition, computational linguistics, forensics, biometrics and bioinformatics. In this particular case we will be looking at the use of pattern recognition software in constructing a software cost prediction model. The two different types of learning algorithms that we can distinguish are the supervised and unsupervised learning methods.

**Supervised learning** refers to the problem of finding structure in labeled training data. This is the usual type of task in machine learning problems: each data point consists of an input vector and an output signal. A supervised learning algorithm analyses the training data and produces an inferred function, which is called a classifier or regression function. If the output is a single discrete class, the result is a classifier. If the output is not a discrete class, but a continuous value, the result is a regression function. Ideally, the inferred function should predict the correct output value for any valid input. This requires the learning algorithm to generalize from the training data to unseen situations in a proper way.
Unsupervised learning refers to the problem of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. Unsupervised learning is more typically used for clustering, outlier detection and pre-processing.

In the case of building a cost estimation model for software, we are primarily looking at a supervised learning task: the costs for completed projects is known. The type of output data can be expressed in man-hours of work or in money. For that reason the type of result that is produced using a supervised learning system is a regression function.

### 3.4 Multilayer perceptron

The multilayer perceptron model is an artificial neural network that maps sets of input data onto a set of appropriate outputs. MLPs consist of multiple layers of nodes in a directed graph, where each layer is fully connected to the next layer. Nodes in the hidden layer are activated by a non-linear activation function. The MLP is an extension of the standard linear perceptron and can distinguish and handle data that is not linearly separable. The ideas and theory of this mathematical model are based on, and inspired by biological neural networks.

A biological neural network is an interconnected collection of neurons, these very simple processing units are the pathways through which information is transmitted in an extremely complex information system. In the human brain, such a neural network consists of approximately $10^{11}$ neurons and $10^{14}$ synapses that connect the individual neurons. Modern computers do not yet have the capabilities to accurately model systems on this massive level. This also points out the limitations that computers have with certain types of hard problems. While humans are very good at solving complex tasks such as language processing, speech recognition, face recognition, use of tools, self-reflecting and learning, machines are better at processing mathematical tasks. Figure 3.1 shows a representation of the neuron.

![Figure 3.1: The incoming input variables $x_1 \ldots x_n$ are multiplied with their corresponding weight factors. The neuron activation function transforms this into an output. Figure from: [21].](image-url)
The biological neural network transmits information in the brain using electrical signals, this combination of an electrical and chemical process is the driving force behind our cognitive awareness [22]. To explain the mathematical working of the (theoretical) multilayer perceptron, we first need to explain the perceptron model:

### 3.4.1 Perceptron

The perceptron was conceived in 1958 by Frank Rosenblatt [23], and later refined by Marvin Minsky and Seymour Papert [24].

The perceptron model consists of two layers: an input and output layer. The multilayered perceptron model contains an additional hidden layer in between the input and output.

In the non-biological concept of the perceptron, the input layer is an \( n \) sized vector of predictor variables. These are presented to the input layer and multiplied by a weight factor and fed into a transfer or activation function.

The resulting value(s) from the transfer function(s) are distributed over the output layer. A representation of the more advanced multilayer perceptron is shown in figure 3.2. In classification tasks where we are looking to match a sample to a categorical target variable there are typically several different possible outputs each representing a single class. In regression analysis with one output variable there is a single neuron in the output layer representing this value.

Neuron output is determined by the activation function over the sum of the weighted inputs. The mathematical notation of neuron output:

\[
\text{n}_{\text{out}} = \varphi\left(\sum_{n=0}^{m} w_n x_n\right) \quad (3.5)
\]

An activation function decides whether or not to fire the neuron upon receiving the input signals. The most simple type of activation function, which is the one found in the perceptron model, is the one based on a threshold \( \theta \). If the threshold is passed the neuron fires, otherwise it does not act. Neural networks expand on this concept by allowing different types of activation functions that transform the real-valued input into a real-valued output.

\[
\varphi(x) = \begin{cases} 
1 & \text{if } x \geq \theta \\
0 & \text{if } x < \theta 
\end{cases}
\]
3.4.2 Neural network

The multilayer perceptron is an extension of the perceptron; there are potentially more hidden layers with more hidden neurons. Additionally the activation function is no longer limited to the threshold function. Figure 3.2 shows the representation of an MLP.

Neural networks are particularly effective for complex regression or classification problems where the relationship between the variables can not be expressed by a simple (linear) mathematical relationship. Some of the positive and negative properties of neural networks are [25]:

- Model free
- Suits complex models
- Can be retrained on new data
- Deals poorly with outliers (overfitting)
- Does not explain its output
- Does not suit small data sets
- Decision process is not visible

The fact that its decision making process is not visible is the most substantial drawback in the context of predictive cost modeling. Convincing someone of the correctness and reliability of a decision making tool is harder when you cannot provide a transparent overview of the reasoning behind the tool. Although neural networks are not particularly suitable to small data sets, previous research has shown that there are opportunities for neural network based approaches to cost estimation on small amounts of data [26]. These methods can provide superior performance compared to standard regression.
MSE = \frac{e_1^2 + e_2^2 + e_3^2 + \ldots + e_n^2}{n} \quad (3.6)

Figure 3.3: MSE is the sum of squared errors divided by the \( n \) samples.

### 3.4.3 Learning

The preceding sections dealt with the perceptron and neural network. But it did not discuss the process of finding out optimal parameters (connection weights) for the network. The general learning algorithm is a fixed loop: sample data is trained, the result is evaluated by an error function, and the parameters are corrected. This continues a fixed amount of time, or a fixed number of iterations, or until the method converges to a solution.

The learning algorithm must minimize an error function \( E(w) \). The traditional strategy is to use the backpropagation algorithm which works by computing the error of the neural net for a given sample, propagating and correcting the error backwards through the network, meanwhile updating the weight vectors. Successfully learning a model implies generalizing from a finite set of inputs to a set of rules that define the model. In a neural network these “rules” are the weights for the connections between the neurons. The exact working of the backpropagation algorithm is explained in chapters 7 and 8 of the book *Neural Networks* [21]. There are more advanced adaptations which mostly improve the speed at which the network is trained. In the implementation discussed in this paper for the purpose of modeling software development productivity we make use of the resilient backpropagation [27] algorithm. Neural network learning algorithms look for the minimum of the error function in weight space using the method of gradient descent [21].

Error function \( E(w) \) is the Mean Square Error. MSE is a measure for the total error of the model on the training samples. Mean squared error is derived by squaring the differences of known true outcomes and their estimates provided by the model, adding those together, and dividing by the number of test samples, and taking the square root of that result. By minimizing and error function the model fits the training data as far as possible. But because we do not fit a relative error it has the danger of fitting itself to the more costly efforts, since they contribute more to the training error. MSE is the standard error function in the Encog neural network learning framework.

### Generalization

Neural network outputs approximately match the target outputs for the given inputs during training. While this is a nice feature its strength lies in being able to approximate results for patterns that were not in the training data. A common misconception is that generalization is always possible when using neural networks, and that it *will* produce
the relationships between variables. Obviously there needs to be some kind of pattern in the data to do any kind of generalization. For any practical application, creating a neural network will also require prior domain knowledge to select appropriate features, and choose the correct settings and learning algorithms. There are three conditions that are usually necessary for good generalization [28].

**Condition one:** Inputs to the network must contain sufficient information about the variable you are trying to predict. Ideally there should be a mathematical function relating the inputs to the correct outputs. One cannot expect a network to learn a nonexistent relationship. It is very well possible that there might not be a direct relationship between any two variables. For example, if you are looking to forecast temperatures and rainfall, a detailed historical database about sunspots is not going make accurate predictions. Temperatures and rainfall may possibly be somehow related to sunspots (or not). But without accurate, detailed and up to date meteorological data trying to make any such predictions is hopeless. Finding the right inputs and collecting enough training data will often take far more time and effort than training the network.

**Condition two:** The problem you are modeling is mostly smooth: a small change in the inputs should produce a small change in the outputs. Very nonsmooth “problems” cannot be generalized by neural nets.

**Condition three:** The number of training cases is sufficiently large and makes up a representative subset of the problem domain. The importance of this condition is related to the fact that there are two different types of generalization: interpolation and extrapolation. Interpolation applies to cases that are surrounded by nearby training cases. Everything outside the edges of the input space in training samples is extrapolation. But also cases inside large “holes” in the training data may effectively require extrapolation. Interpolation can often be done reliably, but extrapolation is notoriously unreliable. Having more data makes needing extrapolation less likely because a new sample is probably inside the range of the training data.

The critical issue in developing a neural network is generalization: how well will the network make predictions for cases that are not in the training set? A network that is not sufficiently complex can fail to detect patterns in a complicated data set, leading to underfitting. A network that is too complex may fit the noise, not just the signal, leading to overfitting. The best way to avoid overfitting is to use lots of noise-free training data, make use of a-priori knowledge, and utilize cross-validation training techniques [28].

### 3.4.4 Confidence measure

Artificial neural networks can be used to construct empirical nonlinear models. Because these network models are not based on a supposed domain specific theory and contain
nonlinearities, their predictions are suspect when extrapolating beyond the range of the original training data. With multiple correlated inputs, it is difficult to recognize exactly when the network is extrapolating. Furthermore, due to non-uniform distribution of the training samples and noise over the domain, the network may have local areas of poor fit even when not extrapolating. The solution is likely to be accurate only for regions sufficiently represented by the training data [29]. This is a key factor that limits the use of these models in practical applications. A source of errors is when completely new data, which is very much unlike the model has seen before, is presented. The two reasons for the inadequacies of the computational model are inaccuracies in the training data, and the limitations of the model. The data variance uncertainty (insufficient and noisy training data), and model variance uncertainty (model limitations), both contribute to the overall prediction uncertainty ($\sigma_p^2$). Model variance determines the confidence in the model estimates ($\sigma_c$), while data variance ($\sigma^2$) determines the prediction intervals of the estimates [30, 31].

$$\sigma_p^2 = \sigma^2 + \sigma_c^2$$

(3.7)

So to estimate a prediction error we need to calculate both model variance uncertainty and data variance uncertainty. They are assumed to be independent, added together they can function as an estimate for a confidence measure. Such a confidence measure helps interpreting results. It may also warn when the predictive model is extrapolating beyond the range of the training data. The proposed model implements a confidence measure by plotting, along with the result, the upper and lower control limits at one standard deviation from the mean as illustrated in figure 3.4.

\[ \text{Figure 3.4: Predictions and the upper and lower control limit set at one standard deviation from the mean. The figure indicates that there is a larger uncertainty in the predictive performance for values smaller than 45 or larger than 60. The red line displays the outcome from the predictive model while the blue line shows the “official” productivity norm.} \]
Neural network ensemble

One of the ways in which it is possible to deal with this issue is to build an ensemble of classifiers. Each slightly different from one another, using different initializations and a different subset of training data, and apply statistics to the different predictions that each of these classifiers produce. Most of the classifiers will produce a very similar prediction: they are trained on (partly) the same data, and predict based on the same variables. But in areas where training data is limited the predictive models are more likely to differ. Since the model is generated and evaluated on mostly the same inputs, it can vary more in sparsely populated areas without having an effect on the training error. Similarly, in data dense areas with a large variance in the target variable, outcomes of the models can differ while having nearly the same training error. When applying a large number of slightly different predictive models on the same problem, their variance in outcome is an indicator of its reliability. Ensembles of neural networks have shown to improve regression performance [32] and can help to avoid overfitting.

The ensemble output \( C_{avg} \) is the average of the outcomes of the \( n \) models:

\[
C_{avg}(x) = \frac{1}{n} \sum_{i=1}^{n} C_i(x)
\]  

The variance over all \( n \) outputs that make up the ensemble classifier can be used to calculate the confidence interval \( \sigma_c^2 \) for input \( x \).

\[
\sigma_c^2(x) = \frac{1}{n-1} \sum_{i=1}^{n} (C_i(x) - C_{avg}(x))^2
\]

A combined ensemble of classifiers for the purpose of improving performance is known as “Bagging” (Bootstrap aggregating) [32]. Bagging combines several learners where each learner uses a bootstrap sample of the original training set. Bagging works better for unstable learners, by averaging over a number of different versions of a model we can attain a better performance and a model less susceptible to generalization. The additional benefit is that you can derive a degree of certainty to each and every possible prediction that you do. Standard deviations from the mean can indicate how classifiers in an ensemble deviate from each other. A high deviation implies that either the training data is spread out, or there is no training data in that input space area. Effectively both of these situations have very little effect on the training error. So it is possible to generate different models that produce different results in these areas while they have a similar training error [33].

Using this bagging approach to training our classifier we can calculate the model variance uncertainty. The data variance uncertainty is also a measure for how far similar projects
\[ \sigma^2(x) = MSE = \frac{\text{SSE}}{n} = \frac{e_1^2 + e_2^2 + e_3^2 + \ldots + e_n^2}{n} \]  

(3.10)

Figure 3.5: The data variance for prediction \( x \) is the sum of squared errors divided by the \( n \) test samples. Effectively this is identical to the MSE.

differ in outcome (costs) even though they have the very same cost drivers. In our problem domain this is a very significant variance. A simple way to approximate data variance uncertainty is to use the total training error. SSE indicates the magnitude of the error produced by the model when evaluated against the training data.

Although more advanced methods exist to calculate the expected prediction variance, such as maximum likelihood methods and separate neural networks for the error measure, a crude estimate based on the total training error is used due to practical considerations.
4. Problem Domain

The compiled B/CAO dataset contains a total of 891 IT projects that have been completed between 2006 and 2013. The information was collected more sporadically in the earlier years, but since 2012 each and every IT project performed in the Belastingdienst that had a scope of more than 100 function points was recorded. Because of several reasons the information is not entirely complete. During 2006 to 2008 a different company (QSM) performed the FPA analysis, and they did not include the total costs of the software development efforts. Since 2009, B/CAO performs their own function point analysis according to Nesma [13]. Nesma is a very specific and exact methodology for doing FPA, but results can still differ between analysts, either because of errors or because of different interpretations. Furthermore the processes and protocols for reporting test results, bugs and incidents were not as refined as they have become later on. So these aspects are harder to compare between the earlier and later years. Most of the information was collected in the QSM tool Datamanager and for the purpose of this research exported to Microsoft Excel in order to do some basic statistical analysis which is reported on in section 4.1.

The information collected is very comprehensive. There are three main sources that have been merged together:

- **Project information** detailing the IT projects, internal costs, external costs, hours spent, start date, end date, primary language, secondary language, % of code in language 1, % of code in language 2, % of modified, deleted and new FP, and defects.

- **Staff expenses** detailing how many hours individuals worked on all these projects, what they spent their time on, their hourly rates, position, role and seniority.

- **System information** detailing the various IT systems in use and in development, their total expenditures, changes, size in lines of code and size in function points.

Chapter 6 describes and motivates the preprocessing steps taken to initially merge this data into a single database. It then details the steps taken to select the appropriate features, construct an additional cost driver, and subsequently prune away the older, irrelevant or unusable projects.
4.1 Project information

Inside the Belastingdienst there are several separate divisions, these divisions are clients of IT projects and owners of the software products or services. Table 4.1 shows the number of IT products or releases that are included in the B/CAO dataset. Obviously the 2013 year is based on the information that was available at the start of this research and only included information up to February. Belastingdienst divisions are: Aangifte, Aanslag, Basisvoorzieningen, Bedrijfsvoering, Dienstverlening, Douane, Gegevens, Invorderingen, Ontvangen, Toeslagen, Toezicht. The total amount of salaries paid out for designing, building and testing all 685 software projects completed in 2009-2013 is over €200,000,000. All other costs, such as hardware, supporting services, travel, upper management, office space, supplies, tools, software licenses, etc, are not included in these figures. It is important to note that these are the expenses for hours that were logged on a particular task. So that implies that salaries for employees who were ill, and did not produce anything but did get paid, are not included in these figures.

\[
\begin{array}{cccccccccccccc}
\text{AG} & \text{AS} & \text{BVZ} & \text{BV} & \text{DV} & \text{DO} & \text{GE} & \text{IV} & \text{OV} & \text{TS} & \text{TZ} & \text{Total} \\
2006 & 2 & 3 & 0 & 0 & 5 & 2 & 3 & 1 & 1 & 8 & 0 & 25 \\
2007 & 7 & 10 & 4 & 3 & 3 & 5 & 7 & 6 & 4 & 11 & 0 & 60 \\
2008 & 14 & 18 & 9 & 4 & 10 & 13 & 14 & 8 & 12 & 12 & 7 & 121 \\
2009 & 8 & 19 & 0 & 3 & 12 & 6 & 14 & 5 & 13 & 11 & 5 & 96 \\
2010 & 11 & 16 & 0 & 8 & 11 & 19 & 27 & 4 & 6 & 10 & 5 & 117 \\
2011 & 17 & 17 & 0 & 9 & 17 & 16 & 49 & 15 & 18 & 18 & 5 & 181 \\
2012 & 30 & 22 & 0 & 7 & 17 & 23 & 69 & 20 & 40 & 16 & 9 & 253 \\
2013 & 1 & 0 & 0 & 3 & 5 & 0 & 13 & 2 & 11 & 0 & 3 & 38 \\
\text{Total} & 90 & 105 & 13 & 37 & 80 & 84 & 196 & 61 & 105 & 86 & 34 & 891 \\
\end{array}
\]

Table 4.1: Belastingdienst divisions and the number of IT projects during 2006-2013. The 2013 year only includes projects upto the month February.

Despite the impressive number of IT projects available in this dataset there are several programming languages severely underrepresented. Table 4.3 points out how the choice for a programming language influences the effort required in building IT systems, and it may be unfeasible to build a cost model for a language where there are only a few samples. These numbers represent the total dataset before any preprocessing has happened. Chapter 6 describes the preprocessing steps and the resulting training set. A further complication which is not illustrated by table 4.3 is the fact that 18.5% actually uses two or more programming languages. The primary programming language is the one in which
most functionality is implemented. Table 4.2 and 4.3 report based on the primary programming language.

<table>
<thead>
<tr>
<th>Year</th>
<th>C++</th>
<th>Cobol</th>
<th>Cool:gen</th>
<th>.Net</th>
<th>IMW</th>
<th>Java</th>
<th>Java-I</th>
<th>PB</th>
<th>SAS</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>0</td>
<td>12</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2007</td>
<td>0</td>
<td>32</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>15</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2008</td>
<td>3</td>
<td>45</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>26</td>
<td>2</td>
<td>16</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>2009</td>
<td>5</td>
<td>37</td>
<td>8</td>
<td>2</td>
<td>10</td>
<td>15</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2010</td>
<td>5</td>
<td>43</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>21</td>
<td>6</td>
<td>14</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>2011</td>
<td>14</td>
<td>69</td>
<td>10</td>
<td>3</td>
<td>7</td>
<td>18</td>
<td>6</td>
<td>16</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>2012</td>
<td>13</td>
<td>97</td>
<td>8</td>
<td>5</td>
<td>13</td>
<td>47</td>
<td>8</td>
<td>28</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>2013</td>
<td>6</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>46</td>
<td>349</td>
<td>63</td>
<td>13</td>
<td>41</td>
<td>150</td>
<td>32</td>
<td>82</td>
<td>77</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 4.2: Number of IT projects split up by main programming language: C++, Cobol, Cool:gen, .Net, IMW, Java, Java-internet, Powerbuilder, SAS and other.

Certain languages are actually development tools which are capable of generating code in a variety of languages. The less well known Cool:gen, Powerbuilder, IMW and SAS are proprietary tools. Cool:gen and Powerbuilder are aimed at building business back-office applications that deal with extensive database communication. SAS (Statistical Analysis System) is an integrated environment of software development tools for statistical analysis and datawarehousing.

4.1.1 Productivity

The productivity of the design, building and implementation phases of the IT projects are measured in function points per hour. During the time this data has been collected, B/CAO has seen a sharp increase in the hourly productivity. Initially, before there was any formal and objectively measurable way of quantifying the work produced by programmers, functional designers, technical designers and testers, it has been shown that it took 31.5 hours to deliver a single FP of functionality. The analysis for 2006 and 2007 had been done retroactively so the employees building the software did not know that their work would be analyzed in this fashion. B/CAO performs significantly better in 2012 at 15.5 hours per function point which is more than a 100% increase compared to 2007; figure 4.1 illustrates this trend. Section 5.2 discusses other potential reasons for this development.

Intuitively it makes sense to assume that higher level programming languages are more effective at producing required functionalities. A higher abstraction level, the availability
of libraries for commonly used functions and automating computing fundamentals (memory management) should result in a smaller code base. The source code uses more natural language and is easier to understand and follow. A lower level programming language on the other hand is more verbose, requiring more lines of code to produce the same result. In practice the required functionalities, and application interfaces usually dictate the programming language that must be used in a particular situation. Table 4.3 indicates the differences in cost per function point for various programming languages. Interestingly, this table does not suggest that higher level programming languages (SAS and Cool:gen) are more effective at producing functionality.

<table>
<thead>
<tr>
<th></th>
<th>C++</th>
<th>Cobol</th>
<th>Cool:gen</th>
<th>IMW</th>
<th>Java</th>
<th>Java-I</th>
<th>PB</th>
<th>SAS</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>14</td>
<td>21</td>
<td>25</td>
<td>24</td>
<td>15</td>
<td>19</td>
<td>12</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>2010</td>
<td>16</td>
<td>20</td>
<td>23</td>
<td>77</td>
<td>16</td>
<td>30</td>
<td>14</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>2011</td>
<td>9</td>
<td>15</td>
<td>19</td>
<td>30</td>
<td>16</td>
<td>23</td>
<td>13</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>2012</td>
<td>8</td>
<td>16</td>
<td>20</td>
<td>26</td>
<td>21</td>
<td>27</td>
<td>15</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 4.3: Hours per function point of functionality in the major programming languages used in B/CAO

A distinctive and clear trend that we can find from the aggregated dataset is that smaller projects tend to be relatively expensive, becoming cheaper up to around 500 to 1000 function points, before they start to get marginally more expensive per FP. A plausible explanation for this phenomenon is that there is a certain amount of programming and administrative overhead before any functionality can be implemented. The technical foundation and graphical interface elements are things which typically take a lot of time to get right, but do not undergo massive changes as functionality is changed or added. So small programs require, relatively, a disproportionately large investment into
Problem Domain

Figure 4.2: Productivity in FP/hour is lower for smaller projects and reaches an optimum around 500-1000 function points. Even larger programs get marginally more expensive again.

non-countable non-functional requirements. Additionally, smaller project efforts suffer in productivity due to a number of fixed start-up procedures. While these initial tasks could be insignificant compared to the overall work involved in lengthy efforts, they are not insignificant for short and small projects.

4.2 Staff expenses

Staff expenses are listed per project, where for each and every person who has worked on it, her hours per task are registered along with hourly rate, position and seniority. It also distinguishes from regular employees, and those who have been temporarily hired from outside companies. For the external temporary employees their seniority and position is not listed.

The cost driver for the average hourly rate is derived from this source of information. Despite the obvious observation that skill and salary do not necessarily correlate it could be a measure for the average weighted skill level of the people who have contributed to the total effort. Another option would have been to take the weighted average seniority level of everyone who worked on it. But the seniority level (junior, medior, senior) is not comparable between roles and the fact that there are some roles which do not define seniority level would further complicate this option.

It is important to note that in the situation of B/CAO there are a large number of externally hired people whose hours cost considerably more than regular employees. This would skew the relationship between hourly rate and productivity. External employees cost significantly more, so if they contribute to the average hourly rate it would result in
a cost driver that is mostly about the ratio of internal to external employees. But we do know the ratio of internal to external project member contribution for (most) projects.

Labor costs for regular employees typically range between 35 and 55 euro per hour. Externally hired people may cost upwards of a hundred euro per hour. Not all individuals’ titles and roles are listed, and there are several cases where it is not clear who was responsible as project manager. For several older projects it was impossible to determine the ratio of internal to external project members. The way that these situations are dealt with is explained in section 4.5.

4.3 System information

A total of 319 IT systems are currently in use or are being developed in the Belastingdienst. Their functional sizes are known, most were counted by function point analysts according to the Nesma counting method. But some which are scheduled to be phased out of maintenance and production, or those that are simply too large and complex, have estimated FP counts by an expert. Other systems may have had their function points estimated based on the total lines of code that make up the system. All 319 Belastingdienst software products contain 299992 function points of functionality, of which 232803 (77.6%) are actually counted, the rest has been estimated. This type of information is important in the cost prediction model because it is expected that the total effort required to expand or modify a computer program is related not only to the size of the modification, but also to the size of the system that is being modified. Highly complex and large information systems could potentially take more work to expand than a very small program, even if the modification is the same size and adds the same amount of functionalities. The benefit of knowing the total system size in predicting future project costs is that you can determine for each and every project in the dataset if is about building a new system, or if it is a new version release. It turns out that for 2012 it took 24 hours to design, program and test a single function point in a new system, while it only took 13 hours on average in an existing system. This makes sense since there is less of a learning curve. In an existing system, the product foundation and technical backbone is operational. And there is existing documentation that only needs to be adjusted or expanded instead of having to start on new functional and technical documentation.

The relative size of the programming effort compared with the overall system size can be derived from this source of data.
4.4 Cost drivers

The cost drivers that will be considered while building the model are:

- Size in function points
- Primary programming language
- Secondary programming language
- % in primary programming language
- Project duration
- Hourly rate of staff
- % of internal employees
- % new function points
- System size
- Function points produced per day

Not all cost drivers are equally reliable. % of internal employees, % new function points, and relative size are incomplete. Section 4.5 goes into more detail about how this issue is dealt with.

All these cost drivers will be used in predicting the number of hours that will be spent. Labor costs and hours are individually connected by the cost driver of average hourly rate. Labor costs and total hours spent are both a good indication of the overall effort required to build a system.

The cost driver function points per day was not explicitly included in the original data, instead it was constructed using the information available about the number of total function points and the turnaround time. Combining these two factors (function points / days) we can build a new feature which could represent the factor of time pressure involved. The consequences and motivations of including this cost driver are further documented in chapter 6.

4.5 Usability

Research question 1 (Does the dataset offer enough data that it can be used to build an accurate cost model for IT projects in the B/CAO organization?) can be answered by investigating the following two issues:

- Size and density of the data and resulting subsets
- Ability to retrieve missing information in the B/CAO dataset


Data size and density

After preprocessing (chapter 6) there are 634 instances left and ten attributes (cost drivers). How can we determine what a good or sufficient minimum sample size is in a model to forecast future observations? A pragmatic definition for ‘good’ is sufficiently rigorous to withstand critical peer reviews. If collecting new samples is without any economic cost, then it would be expected to have a much larger dataset than in the case where it is very costly to obtain additional data. For this research it is unfeasible to collect more data. It would require lots of effort to retrieve more information about IT projects conducted before 2009 at B/CAO. Using data from other companies is incompa-rable and would not result in an accurate cost model for B/CAO, and starting projects to build more data is unreasonable. So usually the sample size is determined by pragmatic considerations.

Some rules of thumb for (multiple) linear regression in the case of independent variables are: 10 samples per attribute [34] and a minimum of 50 samples. Other sources tell to use at least ten times as many training samples as input variables, and a minimum of 5 times as many samples as there are hidden neurons [28]. Obviously these are gross oversimplifications and do not guarantee statistical significance. Additionally, in our predictive model the cost drivers are not independent variables.

Missing information

Despite looking through hundreds of forms it was not possible to determine all of the ten variables for every project. The training set contains 634 samples after preprocessing. Out of these 634, the relative size of a project effort is unknown in just 20 instances (3.2%). Missing values are replaced by the average of this attribute in the preprocessing step (mean imputation).

A ratio of internal to external contribution in hours could not be retrieved in 53 cases (8.4%). Since we only know the eventual total costs and not the precise distribution of work over employees we can only estimate the values in these cases. The average of this attribute is recorded in every case where the real value is unknown.

A percentage of new functionality could be retrieved in only 548 cases (86.4%). These unknowns are replaced by 50%. Further motivation and details on dealing with missing values is provided in chapter 6.
Conclusion

Comparing the size of the dataset to other sources used in studies on productivity in software development [17, 35], it seems that there should be plenty of information to build a usable cost model. The number of projects is relatively large and has been recorded within a small time span. Initial data analysis in section 4.1 has shown that there are certain trends to be found. The enriched database contains little missing data, but in cases with less common programming languages the predictive model may not be as reliable. The number of cost drivers is limited: it would have been interesting and perhaps useful to consider cost drivers such as development process and estimated complexity. An upside to a smaller number of cost drivers is that it becomes possible to generate a simpler that offers insights that are easier to interpret. The size of the dataset is most certainly big enough to generate more accurate cost models than those that are currently being used within B/CAO: QSM-SLIM predicts just 9% within an error margin of 25% [1]. The dataset is complex enough that we should be looking at other factors than just size. Figure 4.3 displays how spread out the distribution of projects is when we only consider size in Nesma function points. So it makes sense to investigate the use of a more complex model.

![Figure 4.3: A scatter plot with all the projects after preprocessing, with the size on the horizontal axis and expenditures in hours on the vertical axis.](image)

4.6 Approach

Question 2 (What is a suitable technique for building a software cost estimation tool specific for the B/CAO organization?) is answered by looking into the purposes of this research and the difficulties that have presented itself within this context.
Dealing with modeling such a difficult problem domain means having to accept a wide margin of error. This is inherent to the task of predicting costs in software development. It would be unrealistic to expect a perfect model. A perfect model which correctly reproduces past performances would generalize poorly seeing how the data is very irregular. In that sense a perfect model in a test environment could be worse in a real-world application than a model with slightly lower performance. An approach to doing cost estimation is to assume that past performance is the best predictor for future events. Using that assumption we focus our efforts on modeling the past, taking care not to overfit the model.

Because previous research has shown that various model based cost estimation methods are not applicable to the environment of B/CAO [1], it is sensible to try a different approach to predicting software development efforts. Neural networks are learning oriented. They start off in a completely random fashion and need to learn the input patterns in order to create meaningful connections between sources of information. While they are generally not suitable to small sets of data, they have shown promise on very similar tasks with even fewer amounts of data [25, 36–40].

Although Belastingdienst / Centrum voor Applicatieontwikkeling & Onderhoud is about building IT systems, its complicated organizational structure does not help in getting software installed on workstations. The IT infrastructure is maintained by Belastingdienst / Centrum voor Infrastructuur en Exploitatie. Getting a new tool installed on B/CAO computers could potentially take a long time. By building a support tool that is entirely web-based and offline, the user is not dependent on local workstation permissions for installing software. Encog itself is a Java applet and does not need to be installed either.

A suitable technique to build a cost estimation tool for B/CAO is to apply pattern recognition algorithms on the available software development data, and create a light-weight web-based graphical interface around this neural network.
5. Productivity in B/CAO

5.1 Environment

B/CAO deals with very specific types of business IT systems that sets it apart from many other IT companies. Its scope is focused on producing large-scale back-office enterprise systems for the various divisions that comprise the Dutch tax office. These systems must be in operation for a long time (decades), but will require continuous updates to implement new and changed requirements. The unusually large portion of programs written in Cobol, or developed in proprietary tools such as Cool:gen, Powerbuilder or SAS reflects the specialized area of IT it is in. And since it is under the responsibility of the Ministry of Finance, it is partly dependent on the demands and requirements brought about by political decisions. When politicians decide upon e.g. tax rates or customs processes, they do not at all consider complicated implementation details. But they do expect the new system to be finished by their deadline.

The public sector is not about making a profit like private sector companies are, instead they are about providing a meaningful service to the community. When comparing studies on productivity or cost estimation for software, one should take this difference in environment and culture into account. Essential differences are that governments do not run the same risk of going bankrupt, so they can afford to be (slightly) inefficient. Public servants have also traditionally had a stronger position in labor law than their colleagues outside of government jobs.

5.2 Organizational changes

Looking at figure 4.1, it becomes evident that there have been influential changes within the organization that have had an impact on the rate at which software is produced. Both the organization and the employees have changed during this time. This section looks into several issues: organizational, technical, financial, and motivational, to identify the changes in these fields from 2009-2012. Primary sources for this information are the B/CAO annual reports [41], a master thesis about implementing Lean at B/CAO [11], an
extensive survey of employee satisfaction performed in April 2013 [42], and internal news articles.

**Function point analysis**

The FP counts used to be done by QSM, but since 2009 it happens internally. There is more control and knowledge about the process, leading to more reliable and comparable results. Belastingdienst programmers and project managers know they will have their work analyzed, so there is less leeway in planning inordinate amounts of time for objectively small tasks. In fact, projects are not meant to be started until there has been an initial Nesma count which is consistent with the proposed planning. This objective FPA count can lead to arguments or even skepticism about the validity of the Nesma counting method, but it still has had an impact on the work process. It is conceivable that productivity is higher because more attention is being paid to the performance of individuals and teams. And because expensive project proposals can be identified earlier, these are now rejected or amended. IBRA is a slightly different counting methodology which is being used nowadays for selected information systems when it is expected that Nesma will produce results that do not correspond with the true effort building these systems. A new counting method called Cosmic is being investigated and evaluated as well.

**Lean**

A study by McKinsey & Company into the B/CAO work processes led to the introduction of Lean [9] in 2010. Lean is a process management methodology and is currently being used throughout the entire organization. It is not a software development methodology, although it is very similar to the Agile conceptual framework. An aspect that Lean has helped to focus on is to pay more attention to assessing and measuring the quality of work and the productivity of people. FPA has also gained further traction in the organization since the implementation of Lean.

The philosophy behind Lean in the context of B/CAO is to transform into a modern and flexible, but more factory-like operation. Some may say that industrialization within IT is not a viable strategy due to the nature of IT which is more akin to craftsmanship requiring specialized skills and abilities. But there seems to be a certain improvement in productivity that can be achieved by streamlining processes, automating routine functions, eliminating redundancy, and making accountability more transparent. To adapt industrialization to the environment of IT, Lean combines factory-style productivity with a more nimble, innovation-focused approach to adapt to rapid change [43]. Applying lean principles can reduce costs by 15 to 25% while shortening development time and improving quality [44].
A common misconception is that Lean is the very same thing as Agile. Agile is a software engineering method and manifest by which code is written, Lean on the other hand is a way by which to organize (for example) this software development process.


**Fewer employees**

The worldwide economic downturn prompted the Dutch government to introduce a number of cutbacks. One way of reducing the yearly national budget deficit was to cut budgets for various public services. The combined number of working hours for all B/CAO employees has decreased by 19.2% between 2009 and 2012. 2.872.000 hours were registered for work performed in 2009, only 2.320.000 hours were spent in 2012 [41].

![Figure 5.1: Drastic budget cuts have led to a significant decrease in the number of people employed. Consequently fewer working hours were spent.](image)

**Expenses**

In line with the budget cutbacks that have led to employing fewer people, total expenditures for B/CAO are 25% lower in 2012 compared to 2009. While B/CAO budget amounted to €236.600.000 in 2009, it cost the Dutch taxpayers €177.500.000 in 2012 [41]. One of the reasons for the additional reduction above the expected decrease due to employing fewer employees, is hiring fewer external programmers and consultants. Hiring external individuals is more costly (per hour) than employing regular employees.
Programming languages

Figure 4.2 provides an overview of the various programming languages that are being used throughout 2009-2013 within B/CAO. B/CAO management has made a conscious decision to gradually phase out old legacy systems which have been built using older programming languages, and to migrate them to new systems built using (more) modern languages. 244 Cobol projects are included in the trainingset after preprocessing. Only seven (2.9%) of those were entirely new systems or migrations. During the same time between 2009 and 2013 we have seen sixteen (12.6%) Java/Java-Internet projects which produced new systems, while the trainingset contains 127 Java/Java-Internet efforts. Cool:gen (2.7% new) and Powerbuilder (0% new) are also clearly being phased out in favor of Java/Java-Internet (12.6% new) and C/C++ (56% new).

Employee satisfaction

To have a more holistic view of the organization we also investigate some other aspects which can affect productivity. There has been a significant improvement since 2007, and it is possible that these changes are not entirely due to technical and financial issues. There are regular monthly assessments and surveys about how employees feel about their work. Together with the average yearly percentage of sick leave this could potentially provide a general idea about how happy staff are in their workplace. Figure 5.4 does not quite present any dramatic changes during 2010-2012. Data from earlier years was not available. But a recent survey by Effectory produced an insightful report [42]. There are significant findings which tell us that B/CAO employees are not entirely happy with their work. Although the aggregated regular monthly assessments do not quite say anything
other than B/CAO is rated at just about a 7, the Effectory report from April 2013 is much more detailed.

The April report has B/CAO staff score an average of 6.7. And it scores worse in nearly all subjects compared to the average for the entire Belastingdienst organization. Working conditions are generally rated as poor, primarily because of the old hardware and software available. People feel that they cannot effectively contribute due to limitations outside of their control (such as old hardware and software). Consequently they consider B/CAO to be inefficient. There would appear to be room for further improvements in productivity by providing employees with more up to date development environments, since they feel that old systems are an important reason for inefficiency.

Furthermore, staff rate the financial rewards as being below what they should earn, although they are more than satisfied with the employment terms other than salaries. The most common suggestion for improvement is to reduce bureaucracy.

**Software quality and customer satisfaction**

B/CAO produces software for internal clients. The tax administration business units have certain demands for new or existing IT systems and are expected to have B/CAO implement these requirements. After the completion of every project the customer is asked about how satisfied she is with the result in terms of functionalities, performance, communication and overall process. These grades are recorded and can be used to rate B/CAO as a whole. Customer satisfaction is also a kind of measure for the quality delivered by the project team.

B/CAO also keeps logs of the number of incidents (bugs, errors) during the testing phase, and during the first month after the system going live. These numbers are an objective
way to measure quality. All these incidents represent additional work that is expended on bringing the systems up to the level they should have been. By dividing the number of incidents by the number of function points, it represents a measure for the quality of the functionality.

Customer satisfaction has improved in 2012, but it still is a relatively low grade at 6.4. There is a significant decrease in the average number of bugs detected in 2012 compared to earlier years. Only 0.026 first month incidents were recorded per function point, compared to 0.044 in 2009. The testing phase has seen fewer incidents as well: 0.37 per function point in 2012, compared to 0.44 in 2009. So while the number of errors that are detected during testing have dropped, there are still even fewer bugs that slip through compared to earlier years.

5.2.1 Conclusion

Research question 3 (What business processes and organizational changes have influenced the increase in productivity since 2009, and how have they impacted the entire organization?) has been answered in the preceding sections. A great number of organizational factors have potentially had an influence on the productivity of software development. Most notably are the structural reorganizations in personnel, and an overall reduction in workforce. The registered improvement in productivity from 2007 to 2008 is primarily because of the introduction of FPA to B/CAO, and because it is no longer done by an external company. Considering the developments over time it is incredibly unlikely that the actual productivity was over 60% higher in 2008 compared to 2007. Different measurement methodologies are the primary cause for the registered improvement.
The sharp drop in hours per function point that is visible in the year 2009 could be because of the formalized approach of FPA within B/CAO. It makes employees aware of the fact that their work and productivity is being measured and monitored. And doing FPA within B/CAO rather than contracting it to an outside company leads to more control over the process, as well as a slightly different methodology which may also partly explain the improvement in productivity.

An evaluation of the introduction of Lean at B/CAO concluded that it had contributed to the rising performance and productivity of the organization. Software quality seems to have improved considerably, both first month defects and test findings dropped sharply. Having fewer bugs during testing also implies there is less work in the initial development phase, because there are less bugs to be fixed right away. The Lean principle of first time right, as well as these findings, underline the importance of avoiding mistakes from the outset of an IT project.

Looking at these figures it appears that the most influential cause for the improved productivity from 2009 and onward is simply the budget cutbacks and layoffs. With considerably fewer hours available, B/CAO has still managed to produce a higher volume of software. Although total hours dropped 19.2% between 2009 and 2012, hours/fp dropped 22.5%. That suggest there was “room” to cut hours.

A second cause is the introduction of Lean. And improvements in software quality, whether or not as a consequence of Lean or FPA, has further accelerated the growth in productivity. A fourth cause is FPA itself, the awareness of being able to measure work volume, and requiring improved project proposals, has led to producing and maintaining more software in less time.
6. Data Preprocessing

To prepare the data that will be used to generate a cost model there are several steps that must be taken:

- Find the missing information and merge the sources together
- Select the features that will be used as cost drivers
- Construct additional features
- Prune away the individual projects that will not be considered when building or evaluating the cost model.

6.1 Enriching and merging

Merging information from various sources introduced certain complications where it turned out that not all details could be found for all projects. Because of the relatively low amount of training samples for the neural network it was decided to use most of the data anyway and approximate the missing values. This section will detail the process and reasoning behind enriching and merging various sources.

The majority of the data had been stored in the QSM tool Datamanager. Domain knowledge about the process of software engineering within B/CAO led to the laborious attempt of retrieving additional sources for additional cost drivers. Three extra cost drivers (making a total of ten) were identified and researched. These three are not 100% complete and were recorded using an error-prone manual process.

The three extra cost drivers are the following:

- **Relative size** of the effort compared to the overall system size. It is calculated by dividing the functional size in Nesma points by the corresponding system size in Nesma points. In the twenty cases (3.2%) where there was no system count it was impossible to determine this value and the average relative size was recorded instead.

- **Percentage of new functionality**: Looking through all the Scope [45] counting session database files we can find out exactly how many new, modified and deleted Nesma
points are awarded. Due to the absence of files for some older projects this cost
driver is incomplete. In 86 of the 634 (13.6%) samples it was impossible to find out
the proportion of new, modified and deleted functionalities. While the total number
of FP is recorded in a database, the new/changed/deleted functionalities had not
been centrally stored.

- In 91.6% the percentage of total effort by internal employees could be retrieved from
  the staff and salaries database. But for 53 of the older projects this information was
  unavailable. In these cases the average value was recorded.

By applying mean imputation rather than listwise deletion we do introduce some data
pollution. But the alternative would have led to a lower sample size, and a potential bias
introduced by deleting projects with missing attributes. Therefor it was decided to not
discard any of the data where an attribute was missing and apply mean imputation.

The argument for including the first additional cost driver is that relative size is expected
to influence the effort to build and test this new system. Because B/CAO maintains a
list of all Belastingdienst software services and their size in terms of function points and
lines of code, it is possible to find out about the relative size of each software project.
Although some are about building an entirely new system, the overwhelming majority are
(small) release versions that add, modify or fix some functionality. Since there was only
an up-to-date list of the IT systems it was impossible to accurately determine the relative
size for all instances. Past updates and software versions have influenced the function
point counts, and it is no longer possible to find out about the exact size of the overall
system at the time that that version was produced. So instead it has been estimated
using the up-to-date list which should still give a good approximation of the relative size,
especially for the projects completed more recently. These percentages are rounded up to
whole numbers.

The rationale for investigating the proportion of new functionality is that it adds im-
portant information about a software engineering effort. Nesma FPA awards points to
added, changed and deleted functionality. They all add up equally, but instinctively there
is a substantial difference. Removing functionality should in many cases be easier and
cheaper than adding new options. Changing functionality could be about configuration,
and is probably cheaper than programming new systems, yet more expensive than remov-
ing code. It is expected that the different kinds of function points influence productivity
differently, and by adding this information to the training set we can find out. Percentages
are rounded to whole numbers. There is very little deleted functionality in the training
samples, so that aspect is not considered. By only storing the added functionality as a
cost driver we also implicitly store the modified functionality (100 - added functionality).
An average hourly rate gives a very rough idea of the seniority level of a project team, but it quickly turned out that there was another more important influence on the attribute of hourly rate. The amount of externally hired people working on a project skewed the average hourly rate. But together, they do provide information about the level of seniority (as well as the ratio of internal to external employees).

\section*{6.2 Feature selection}

An issue in pattern analysis is feature selection, in the case of cost estimation the features are (potential) cost drivers. Especially in high-dimensional datasets there is a risk of deteriorating performance of data-mining algorithms. The decision on whether or not to actually use these features as cost drivers in the eventual model is based on the added information they provide in accurately predicting the target variable: hours. Another factor that influences the decision is the required or maximum complexity of the generated model. Including additional features that do not add relevant information will only reduce overall performance. So instead of using all variables, we want to select a subset of features which together are most effective for the purpose of predicting costs. Models with fewer features can generalize better on unseen data as the model has not been overfitted to the training set. The need to select features in order to reduce the complexity of the system leads to lower storage needs, lower computational needs in building the discriminant system, and lower computing power in analyzing new samples. A model with fewer variables also results in a system that is more easily interpretable.

An initial selection of cost drivers out of the starting data was based on existing domain knowledge. Project manager seniority was initially included but it turned out that it was not a suitable cost driver. It was retrieved manually by searching through hundreds of evaluation forms. And these evaluation forms did not list the correct manager for all projects. So the information was unreliable from the outset, but the main reason for excluding it was the poorer predictive performance of the cost model. It turned out that including this factor meant the trained neural network would not generalize well. There were several combinations of programming languages and seniority (junior/medior/senior) that were not included at all in the training set. Furthermore most junior managers were given small projects, none larger than 700 function points.

Having a model that can generalize well beyond the training data is crucial. Including the project manager variable did contribute to improving the performance of the predictive model on training data, but generalization suffered.
6.3 Feature construction

Besides reducing the number of features, another approach is to artificially increase the number of features by either including more data or by combining features. Doing this can result in a more suitable dataset that can be learned more effectively by neural networks. In theory a neural network should be able to learn any non-linear function. Thus it would not make sense to add redundant information. But in practice it can be helpful to include the right logical combination of cost drivers to a dataset. Research has shown that classification performance (for neural networks and other methods) can improve by domain knowledge guided feature construction [46].

Turnaround time and function points are strongly dependent on each other. Small projects last shorter than large projects. And each of them separately do not necessarily convey any kind of information with regards to the time pressure on employees. But combining these two could provide us with some additional knowledge about that particular development process. In this case we can find out about the number of function points produced per day. That daily productivity could indicate time pressure, difficulty of the project, or the size of the project team. None of these variables are explicitly recorded in the existing dataset.

A large number of function points per day would tell us that the time pressure is high, and/or the project team is large, and/or development is simple. These three are combined in a single value and make up the last cost driver. Preferably the time pressure, team size, and complexity would have been recorded separately. Having it combined into a single value means we could lose information when effects are canceled out against each other (high time pressure and a small team may result in the same FP/day as low time pressure and a large team).

It turned out that including this cost driver improved performance considerably compared to the method without.

6.4 Pruning projects

An essential part of pre-processing is removing errors and other noisy data. Detecting which samples are, and which are not noise is a difficult problem. Similarly to “noisy” features, it is possible that an entire project is an anomaly in the collected dataset. It would be unwise to consider this example while training your model, since it would result in a model which is unrepresentative of the real implementation environment. Therefore we need to remove samples which are known to be completely unrepresentative of the software engineering process in B/CAO. In addition we apply a crude filter to remove
projects that lie beyond the range of productivity that we can expect to be able to predict.

But first of all the greatest reduction is achieved by removing the years 2006-2008 from the dataset (206 instances). There is no cost and staff information about these projects so these cannot be used in building our model. Additionally we have seen that these years are not representative of the current situation in B/CAO since productivity is drastically different, and numerous organizational changes have occurred. Five other newer projects that also lack cost information and two more which dealt exclusively with deleted functionality were removed. This leaves us with 678 samples.

All 25 efforts where either the client was an external company, or development was completely contracted to external developers, are discarded. Since B/CAO is not looking for a cost model for external developers this data is mostly irrelevant. All efforts that were larger than 2000 function points are removed.

A number of outliers are pruned where productivity was higher than 2 hours/FP, or those that cost more than 5000 €/FP. Filtering based on the target variable (productivity) is dangerous because it could introduce a bias to your predictor. After these actions the resulting training set is comprised of 634 samples.

6.5 Normalization

Encog [47], the machine learning framework, will automatically normalize the input features between 0 and 1. Equilateral normalization [48] is applied to the input classes (programming languages). The normalization step helps to improve performance.
7. Proposed Model

7.1 Motivation

Requirements for a new B/CAO cost model used in predicting software development costs are:

- It can make accurate predictions, performing better than the currently used cost model
- Its output should be clear and easy to interpret
- The impact of a cost driver can be visualized graphically
- The code is maintainable
- The model can be retrained using new data

For a tool that is used in producing estimates it is crucial that it can convince the user of its reliability and usefulness. Because of these requirements the proposed model is generated by an open source machine learning framework. All code that produces the estimates is available and can be checked for errors. The working and influence of the cost drivers can be demonstrated graphically through a web-based interface.

7.2 Implementation

Encog [47] is a machine learning framework which supports a variety of advanced machine learning algorithms. There are Java, C++, C# and Javascript frameworks for Encog. The Java version is the most up-to-date and supports most features. Therefore the Java Encog program was used in training and generating the neural network used as basis for the predictive model. Other machine learning algorithms such as Support Vector Machines are also available in Encog.

Other researchers have applied neural network learning algorithms to the same issue of predictive cost modeling and cost estimation in software development [25, 36–40], as well as in manufacturing costs [26]. Their findings are slightly mixed but generally conclude that
neural network approaches are superior to standard regression analysis. These findings and other research on cost estimation are the main reasons for taking a similar approach.

The neural network with twenty input units, one output unit, and 61 hidden units in the hidden layer was trained using the resilient propagation algorithm. Learning stopped when the Encog trainer reached a training error of two percent. A bagging approach with an ensemble of fifty trained neural networks make up the eventual predictor. The average of these predictions is calculated and graphically reported to the user. The neural networks are exported to a javascript implementation which connects it to a simple user interface. Benefits of an ensemble classifier are: less susceptible to overfitting, performs better on unseen samples, and produces a more realistic error.

There are twenty input units due to the available input cost drivers and the preprocessing steps that expand the non-numerical cost drivers into a combination of unique sets that each represent a single class value. There are four options for the secondary programming language (none, cobol, java, other), resulting in three input units for the neural network. Together with the eight ordinal cost drivers, and the nine inputs that represent the primary programming language this makes for twenty in total.

There is obviously a single output unit which signals the expenditures in hours. Determining the optimal number of hidden units is very hard and there is no clear methodology for doing this other than searching and comparing various results. Encog suggests a default amount of hidden units based on the number of inputs, in this case it suggested 61. Investigating the optimal configuration of hidden layers and hidden neurons would probably improve performance of the predictive model even further.

7.2.1 Interface

The interface is based on a combination of the Bootstrap front-end framework for web development by Twitter, the Google Chart Javascript library and the Encog [47] neural network implementation. A simple web-based form view allows the user to enter the values for all the available cost drivers, and the cost estimation can be calculated. In order to provide additional insights into the effects of various cost drivers it is possible to plot a range of values for a cost driver.

Besides the cost drivers there are additional options to change the plot resolution (low, medium, high), and to change the plot size (small, medium, high).
7.3 Instructions

The main use case is to compute an estimate for the costs of software development: enter all the required cost drivers and click on Bereken (Compute) to produce a cost estimate. To plot the results you first have to enter the cost drivers. Productivity for the current combination of cost drivers in plotted along the y-axis. By altering a single cost driver while keeping the rest equal it is possible to compute and compare the effect of that single variable. Outputs are shown in hours per function point in each of the graphs, as well as total number of hours.

A secondary use case is to reflect on, and evaluate software development efforts using the cost estimation tool in a retrospective analysis. It could potentially help to explain deviations from the artificial norm by comparing it to similar projects that were completed in the past.

The tool can give an indication how productivity changes as cost drivers are altered. This is done by taking the cost drivers as entered by the user as base. Each plot shows how productivity changes as one single cost driver is altered. The alteration of that single aspect is plotted along the x-axis with productivity on the y-axis.

Estimates from this predictive model are based on what has been observed before, they indicate if results are consistent with previous performance.

Most cost drivers have certain upper and lower bounds:

- **Function points** - 25-2000 The number of function points counted using the Nesma counting rules
- **% new function points** - 0-100 Percentage of new functionality
- **Project duration** - 25-700 The number of calendar days between the start and end of the entire project.
- **Hourly rate** - 30-100 The weighted average hourly salary in euros of project members throughout the duration of the entire project.
- **% of internal employees** - 0-100 Percentage of internal project members’ contribution to the overall effort.
- **System size** - 1-FP The Nesma size of the eventual product.
- **Primary programming language** - Cobol / Java / Cool:gen / C++ / Java-internet / IMW / SAS / SAS-Base / Powerbuilder / Other Primary programming language
- **Secondary programming language** - None / Cobol / Java / Other Secondary programming language
- **% in primary programming language** - 50-100 Percentage of the overall size of the project programmed in the primary programming language
In areas where the model is sparsely populated by its training data, it will generate progressively worse predictions. Examples of such situations are for example a 2000 FP Powerbuilder project that has a duration of 50 days. The training data does not contain any Powerbuilder projects of that size and there is no sample project where productivity was higher than 11 function points per day. Evidently, the predictive model is not suitable for such incredibly unlikely scenarios and will not produce any predictions. Additionally the model will indicate where there is a large variance in predictions. A high variance implies that predictions in that region of input-space are uncertain.

Combinations of cost drivers that are outside the observed range of that variable within are not plotted. Computing the convex hull of the input space was another option, but this proved to be very limiting for less common programming languages and in cases where two languages are used.

The end-user of the cost model must consider and take into account real-world constraints that limit the effectiveness of a cost prediction. A considerable drawback and pitfall is that it is still possible to generate predictions that can never be realized. Unlikely or impossible combinations of cost drivers might theoretically end up in a cost-effective solution but may be practically unworkable.

7.4 Maintainability

By using open source software and code (Encog, Bootstrap and the Google Graph library) the tool has become far more maintainable. The quality and reliability of the open source code is far higher than what could have been achieved within the limited amount of time by a single programmer. The size of the Javascript which handles the connection between the neural network, forms and graphs is only 850 lines of code. The rest of the tool is updated and maintained by the open source community and HeatonResearch (Encog), Twitter (Bootstrap) and Google (Chart api).

Most effort in maintaining the tool will be in keeping the cost model up-to-date. Adding new projects to the model will take the following steps:

- Evaluate the suitability of the project in the cost model. Leave out unrepresentative (external) efforts. Remove outliers (productivity per FP too high or too low, costs per FP too high or too low).
- Add suitable projects to the training set.
- (Optionally) Remove old unrepresentative projects from the existing dataset.
- Generate new models using Encog and the specified parameter settings. See Appendix A.
• Export the generated weights into the cost model implementation. See Appendix A.

7.5 Analysis

The analysis focuses on the productivity in software development, B/CAO is particularly interested in how the various cost drivers determine the productivity in software development. Productivity is measured in hours per function point. We have already found that there is a clear connection between the size of software projects and the productivity per hour. In figure 4.1 it shows that there is an optimum between 500 and 1000 function points. The generated model confirms these observations. Besides confirmation it also offers some detail about the influence of the individual and joint cost drivers on productivity. Two different types of cost drivers are considered: variable cost drivers and fixed cost drivers. Although the tool and model can compute any combination of inputs, such a scenario is not realistic in practice. Certain cost drivers are fixed for a particular type of project while others can be steered. Programming languages and system size are fixed. The environment decides these issues, and it is mostly impossible to deviate away from it. Other cost drivers such as function points, turnaround time, hourly rate and internal:external ratio could potentially be altered if that would be more likely to produce better results.

When interpreting these figures it is highly important to realize that these patterns are derived from past performance in software engineering in B/CAO. Errors, incompleteness and coincidences in the collected data may be partly or wholly responsible for the trends that are uncovered. As we know from disclaimers on financial products: past results do not offer any guarantees for future performance, so to all intents and purposes the user would be well-advised to consult the findings from any predictive model with an expert.

All of these analyses are based on the aggregated information of 634 software development efforts. This combined data has produced a computational model which has led to the patterns that are being presented in this chapter. None of the conclusions pertain to a single project alone.

7.5.1 Method

Since the model is very complicated and all cost drivers are interconnected it is hard to give a single absolute trend for all possibilities for a particular cost driver. Cobol projects are different from Java projects, migrations are different from patches, short efforts are different from longer projects. The complicated interplay between these variables is caught
in the model, but it implies that we cannot provide a single trend for all possibilities. Therefore we have simplified the analysis by referring to four typical example projects, and by showing overall average trends per programming language per cost driver. The four examples projects are:

- **1** - Cobol, Size: 500 FP, System: 1500FP, 370 days, 10% new, €50/hour
- **2** - Java, Size: 100 FP, System: 3500FP, 150 days, 0% new, €40/hour
- **3** - C++, Size: 300 FP, System: 300FP, 250 days, 50% new, €65/hour, 15% external
- **4** - 70% Cobol, 30% Java, Size: 100 FP, System: 2500FP, 200 days, 40% new, €70/hour, 10% external

Besides the four examples, all of the 2009-2013 projects used in the training data are split up by primary language and their cost drivers are input into the predictive model. The averages of the generated outputs for that particular programming language and specific cost driver are shown to present the overall trends.

By computing the entire range of the cost driver for all projects and averaging over all these predictions we do not calculate the actual expected average productivity. The method implies that this average is computed from all projects where one cost driver is fixed to a certain value, but the other cost drivers have not changed. As such the results from the overall graphs that display the trendlines for each programming language are not predictions of productivity for new projects. The trendlines indicate the direction in which productivity develops.

The analysis focuses on the cost drivers. We are interested in finding out about the relationships between these factors and the overall costs and productivity. By looking at individual cost drivers we attempt to give a simple indication of the way this single factor changes productivity in software development at B/CAO.
7.5.2 Functional size

Functional size is a major influence in costs and productivity. It is a measure for the size and number of functionalities in a computer program. Within B/CAO Nesma is the leading function point analysis method. We have already seen that smaller projects are less efficient. The cost model generated out of the dataset confirms this trend: as functional size increases, so does average productivity.

![Diagram](image)

**Figure 7.1:** Average productivity trendlines for all programming languages. These trendlines were generated by computing predictions for all training samples while adjusting the parameter of functional size, all other parameters remain constant. Observations are consistent with earlier analysis which states that productivity improves as projects become larger in functional size.

The optimum for all possible projects is not fixed: it depends on the other parameters and in particular the primary programming language. In general the optimum can be found between 500 and 1500 function points. Below 50 FP, productivity is drastically lower. Beyond the optimum, development efforts will gradually become more expensive.
per function point. The exact optimum can be calculated and plotted case-by-case using the B/CAO cost estimation tool.

The general trend in functional size is that larger projects cost less per function point, which is consistent with the analysis in 4.1. The most relevant portion of the graph is the part between 0 and 500 function points. Beyond that range there is very little training data. Capers Jones has shown that very large software systems are more likely to be canceled, run over budget or take too long [49]. The B/CAO dataset does not contain canceled projects, and does not contain very large (>2000 FP) systems. Jones his research suggests that even larger projects run a greater risk of being canceled and may not actually be more cost-effective.

![Graphs of Project 1, Project 2, Project 3, Project 4]

**Figure 7.2:** Our individual examples projects show similar trends in productivity. Project three had a system size of 300 FP, the model only calculates predictions upto that point. Project four shows peculiar behavior after 600 FP. Seeing how the error margins (gray lines) expand at that point this is likely to be an area where there is no training data for that combination of cost drivers.
7.5.3 Duration

The factor of time is represented in the model by the number of calendar days between the start and end-date. So this number is not equal to the actual working days. A pitfall in using this measure of calendar days is that it can misrepresent the actual time pressure and the actual working days. For example: a project that has started on the July 1 and lasts up to September 1 contains fewer productive working days than one which starts on September 1 and lasts until November 1. The summer holiday season means that there are fewer people available and they will work fewer hours per week. These implications are not corrected against in the cost model.

![Productivity and duration](image1)

![Productivity and duration](image2)

**Figure 7.3:** Generally taking less time is better, except in extreme cases where $FP/day$ is extraordinarily high.

A key conclusion is that duration is a factor which is mostly a consequence of the other properties of an IT development project. Running over time is an indicator of a poorly managed, poorly implemented, or poorly defined project; thus running a greater risk of running over budget. In situations where it turns out that there are unforeseen implementation problems, specifications change halfway, or other influences negatively impact the process, a project will take more time than planned and costs rise. Combined with
the observations that cheap and easy projects take little time, while hard and expensive projects take more time, we see a very clear trend line across the entire population of projects: building a system in less time is cheaper than taking more time. The crux is interpreting these results and matching them with the reality where oftentimes outside circumstances dictate what is, and is not possible. Although these figures suggest that the best way to improve productivity is by shortening project duration, such options are often unrealistic.

The plots imply that people will take more time for the same amount of work if they are given more time. Parkinson’s principle: *Work expands to fill the available volume* suggests the same phenomenon. However it would be incorrect to draw this conclusion based on the available data. The cost driver of time is not based on the planned turnaround time, nor is it based on the planned working hours. Instead it is merely the time it effectively took to complete software development and deliver the product.

In graphs 7.3 there is a very pronounced drop in productivity as projects take more time. Figure 4.3 clearly tells that there are significantly more smaller projects than larger projects. Similarly, there are more short than long projects. A significant portion of the graph on duration and productivity is based on extrapolation. What it does manage to convey is that it is not sensible to let projects take longer than needed.

**Figure 7.4:** Findings in the example projects are consistent with earlier analysis. The confidence measure provided by the predictive model gives an indication of which predictions are accurate and where there is a great deal of uncertainty. Expanding error margins tell us that the model can no longer accurately estimate required effort.
7.5.4 Average hourly rate

When considering that very few projects are completed at an average salary higher than 70 €/hour, most programming languages show that productivity drops as salary increases. Only C++, Java-internet, and IMW go against this trend. It is counterintuitive that more expensive employees are less productive. But there are plenty of possible reasons that could be an explanation. For example: senior employees are assigned to harder, time critical assignments that involve building new systems rather than doing modifications. In any case, there does not seem to be a very strong effect on productivity.
Figure 7.6: Our four example projects show varying developments, which do not necessarily match with the overall trends as pictured above. There does not seem to be a very strong connection between hourly rate and productivity (especially within the most common range of 40€/hour - 70€/hour). However there is no direct relation that always implies that a more senior project team is more effective and more efficient at building software. The wide error margins for projects completed at a rate higher than 70€/hour indicate that there are very few samples in the training set that fall within that category.
7.5.5 % External employees

The proportion of internal to external employees and their contribution in hours is strongly correlated with the average hourly rate. In this analysis we only look at the individual effect of the cost drivers. In practice there are very few projects where less than 80% is done by regular employees. The general trend suggests that productivity suffers very slightly when regular employees are replaced by external programmers.

Some considerations that could explain this phenomenon is that they are placed on more challenging assignments, or they develop IT systems in languages which are inherently more expensive. Alternatively it could simply be the case that they are not yet familiar with processes, tools and software products which are being developed at B/CAO.
Figure 7.8: Productivity improves slightly as a greater portion of the project team are regular employees as opposed to externally hired. The difference is marginal though, especially considering how there are very few examples where over 20% is hired externally.

When separated by programming language the effect is more substantial compared to when all results are averaged from all projects. For some programming languages there is a slight positive correlation, while in others there is a negative correlation.
By distinguishing between new and modified functionality we can tell that not all Nesma function points are quite equal in terms of the costs associated with them. The general trend that can be distilled from the model is that building new functionality is more expensive and will take more time. Modifying existing code is considerably cheaper.

Though not all programming languages follow these rules, SAS, IMW, Powerbuilder and Java-Internet show a flat or positive development in productivity when going from 100% modification to 100% new.

In the individual cases where we look at four projects in particular we can see the same upwards trend as was observed in the average trends. The respective programming languages are Cobol, Java, C++ and Cobol/Java. For these languages it takes more time to
produce new functionality than it does to alter existing software.

**Figure 7.10:** The example projects confirm the general trends for their respective programming languages.
7.5.7 Multiple programming languages

Project 4 is the only of our four examples which deals with multiple programming languages. It is a mix of Cobol and Java, we already know that Cobol is generally cheaper than Java (shown in table 4.3). So it makes sense that costs increase as a larger portion is programmed in Java. But also the integration of multiple languages into a single system brings about additional challenges that makes these types of systems more expensive. Because even when the primary and secondary languages are reversed, productivity decreased although there is a greater portion of the “cheaper” languages. The second figure illustrates that very effect: with secondary and primary languages reversed the same downwards trend is visible as in the first graph.

![Graphs showing the relationship between Cobol and Java percentages and productivity]

**Figure 7.11:** Multiple programming languages within a single project usually results in more expensive software.

Interestingly the predictive model does not compute similar values for the 50% mark, which should be identical. The neural network is limited and does not recognize that these situations are equal. While Cobol is a cheaper language, implementations become more expensive still as there is a more even mix between primary and secondary programming language. Apparently integration between the components built in different languages is harder or more time consuming.

When there is a larger difference in standard productivity between the two languages, that may overpower the effect of additional multi-language integration. IMW required 26 hours to produce a single function point in 2012 (on average). While Cobol required 16. Combining these two languages in a single computer system shows that, regardless of the exact proportion, less of the more expensive language results in cheaper software.

So these trends are highly dependent on the specific programming languages. In general we can conclude that projects with multiple languages are more costly compared to using a single programming language. Having a more even amount of functionality programmed in both languages will require more integration and end up being more costly. But the main cost driver is still the programming languages that are being used.
7.6 Performance

Training a neural network by bootstrapping the data makes for a more honest evaluation of the model on unseen new samples. But it also reduces regression performance on the training set. The final model produces an $R^2$ of 0.789. Average predictions are 61.9% off of the real cost. Smaller projects cannot be as accurately predicted as the larger ones, and this abundance of tiny projects propels the average error upwards. 48.9% of the training samples are predicted within 25% of the actual costs.

Figure 7.12 clearly presents that the classifier does poorly on smaller efforts. A plausible reason for doing better on larger projects is the choice for the error function in the neural network. MSE takes the average of the squared error, and thus weighs lengthier projects heavier. Overestimating a 10000 hour project by 1%, contributes as much as overestimating a 100 hour project by 100%. Therefore the network puts more emphasis on fitting larger projects. Using a relative error rather than an absolute error such as MSE could have prevented this problem. Alternatively predicting $\log(\text{hours})$ instead of the total hours would have mitigated this effect.

Research question 4 (Does the new software cost estimation tool produce better results predicting the costs of software projects in the B/CAO organization than other tools, and QSM SLIM in particular?) can be answered with yes. QSM SLIM predicted merely 9% within an error margin of 25%, while the neural network based approach reaches an $R^2$ of 0.789, and predicts 49% within 25%.
Previous research had already looked into the predictive performance of a number of other models [1], but evaluated them only on a very limited and handpicked subset of the 2010 and 2011 data. This subset only included projects larger than 100 function points, while this paper considers all sizes. This aspect alone accounts for a substantially lower reported predictive performance of the neural network model, while under similar circumstances with the very same data it would perform far better.

Masselink reported findings and performance using the Mean Magnitude of Relative Error (MMRE). MMRE, despite being a popular choice for software cost estimation, has been shown to be a poor evaluation criterion [50]. Pred(25) is a much better choice.

<table>
<thead>
<tr>
<th></th>
<th>MMRE</th>
<th>Pred(25)</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural network</td>
<td>0.618</td>
<td>48.9%</td>
<td>0.789</td>
</tr>
<tr>
<td>QSM SLIM</td>
<td>4.26</td>
<td>9%</td>
<td></td>
</tr>
<tr>
<td>COCOMO II</td>
<td>0.3</td>
<td>46%</td>
<td></td>
</tr>
<tr>
<td>SEER-SEM</td>
<td>1.17</td>
<td>29%</td>
<td></td>
</tr>
<tr>
<td>Cobra</td>
<td>0.7</td>
<td>22%</td>
<td></td>
</tr>
<tr>
<td>KnowledgePLAN</td>
<td>2.68</td>
<td>11%</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Predictive models and their regression performance. The estimates based on models other than the neural network are evaluated on a limited subset of the 2010 and 2011 data [1]. The neural network is superior to the others seeing how it predicts better on a harder, larger and more varied set of data. The column under Pred(25) is what percentage can be estimated within 25% of the true value.

When looking at the performance of the neural network model for each programming language we can see some differences. Here we only look at the primary programming languages. Certain languages are much easier to predict than others. Cool:gen, SAS-Base, Other, C++ and SAS are relatively easy to predict while IMW and Java-Internet are much harder.
### Table 7.2: Neural network model regression performance by programming language.

<table>
<thead>
<tr>
<th>Language</th>
<th>MMRE</th>
<th>$R^2$</th>
<th>Pred(25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0.618</td>
<td>0.789</td>
<td>48.9%</td>
</tr>
<tr>
<td>Cobol</td>
<td>0.888</td>
<td>0.702</td>
<td>44.2%</td>
</tr>
<tr>
<td>Java</td>
<td>0.395</td>
<td>0.714</td>
<td>52.9%</td>
</tr>
<tr>
<td>Cool:gen</td>
<td>0.173</td>
<td>0.862</td>
<td>78.8%</td>
</tr>
<tr>
<td>Java-Internet</td>
<td>0.401</td>
<td>0.79</td>
<td>44%</td>
</tr>
<tr>
<td>C++</td>
<td>0.3</td>
<td>0.540</td>
<td>58.3%</td>
</tr>
<tr>
<td>SAS</td>
<td>0.484</td>
<td>0.803</td>
<td>50%</td>
</tr>
<tr>
<td>SAS-Base</td>
<td>0.23</td>
<td>0.691</td>
<td>70%</td>
</tr>
<tr>
<td>IMW</td>
<td>1.03</td>
<td>0.686</td>
<td>34.5%</td>
</tr>
<tr>
<td>Powerbuilder</td>
<td>0.519</td>
<td>0.725</td>
<td>41.5%</td>
</tr>
<tr>
<td>Other</td>
<td>0.272</td>
<td>0.737</td>
<td>63.2%</td>
</tr>
</tbody>
</table>

Cool:gen is easier to predict, Cool:gen usually involves extensive and lengthy development projects. IMW, which is hard to estimate accurately, is usually about relatively small systems (in function points).
8. Discussion

The initial reason for researching cost estimation models was the realization that QSM-SLIM did not produce satisfactory results. The subsequent research by Inge Masselink concluded that other models were also unsuitable. This paper reports on the findings of a new neural network based model. So how does it compare to the currently used QSM model, and in what ways can it benefit B/CAO?

8.1 Comparison with QSM SLIM

Compared to the QSM model [18] the improved neural network based cost model brings a number of advantages:

- First and foremost is the increased predictive performance in planning IT efforts. A better estimate outlines the probable expenditures over time and immediately highlights high-risk projects. Detecting problems earlier, or even before a project has started can save resources.
- The currently used QSM model for cost estimation has high licensing costs, unlike the proposed model.
- It uses a graphical interface that can plot a range of cost estimations.
- It produces trendlines for the effects of individual cost drivers.
- Based on open source toolkits so that it can be freely adjusted to fit B/CAO specific needs and requirements.

Because it is based on empirical observations, it compares with the past while the QSM estimate is all about a theoretical idea how software development should be. It can compare estimated future performance with the required norm as well as with the historical data.

The downsides of this method compared to the QSM-SLIM technique are:

- No support
• No integration with other reporting and database tools  
• “Black box” approach  

8.2 Opportunities

Possibilities that accurate cost estimation can offer to controlling the process of building software are plentiful. Some examples of actions where new and better insights could lead to advantages are the following:

• Adjust the planning to allocate efforts in a way that is optimal according to the model parameters.  
• Plan designing, programming and testing efforts based on the estimated costs and the staff available.

Planning is usually based on proposals put forward by the project manager. Her expertise, skill and experience lead her to decide on a planning in which it should be possible to complete the requirements. An initial planning is sufficient in predicting the overall costs if everything goes according to plan. Comparing these figures against an initial estimate produced by an automated tool should give an indication if the planning is realistic. Further discussions about allocating resources can be helped or steered by the estimates produced.

Improvements in productivity can be gained by steering B/CAO work processes to better fit an optimal configuration of resources according to the cost model.

Theoretically there could be possibilities where it makes sense to change parameters such as functional size, duration or staffing to obtain similar results with fewer expenses. Steering an effort into the optimal direction is more likely to result in higher productivity. An example in the B/CAO environment is the scheduled delivery of maintenance updates to the ABS (Aanslag Belasting Systeem) system. This is one of the largest IT systems in the Belastingdienst and processes tax returns. Usually three releases are scheduled for each and every year and comprise of around 1500 function points each. There seems to be a steady inflow of new or changing requirements, primarily because of political decisions about tax reforms. Together with additional non-critical functionality, it provides a consistent amount of work. Release sizes for new versions usually amount to around 1500 function points. But if it turns out that another size combined with another release cycle is more productive, there is a real incentive to steer it into that direction. Numerous other IT systems deal with a similar scheduled release cycle where this business case might be applicable.
An organizational change within the process of developing software is to make people aware of their individual and combined performance to help them push towards targets. The burn down chart typically used in agile programming environments captures the essence of this principle. Combined with accurate cost estimations these charts are far more effective since they provide a realistic goal that is supported by historical data. A realistic goal makes it possible for management to hold people responsible for their performance, since performance can be measured to be better or worse than what would have been expected.

A predictive model can also help to explain situations afterwards, for example to explain why productivity was lower or higher than the benchmark. Project managers are evaluated on their performance which is determined to a certain extent by metrics such as function points per hour for the entire team. Additional knowledge about the process of software engineering and its cost drivers will help to evaluate them more fairly. Current B/CAO productivity norms are entirely fixed, and depend only on programming language. But this research findings show that other parameters do have a considerable impact on productivity, and should also be considered when deciding on a productivity norm.

B/CAO can optimize productivity in software development (research question 5) by planning a more appropriate amount of resources on tasks.

8.3 Limitations

A theoretical model, even if generated out of real-world data, is still only a limited representation of real-world processes. The performance of the neural network, which still does not even manage to predict half of the cases within an error margin of 25%, painfully underlines its limitations.

Some considerations that must be taken into account are:

- **Missing data** - Section 4.5 details what cost drivers are incomplete and what process was used in retrieving some of this information. A noisy and incomplete dataset leads to errors in the eventual model.
- **Missing cost drivers** - This paper has pointed out that it would have been preferable to have access to more data. Additional information could further improve the model.
- **Limited effect of cost drivers** - Ratio of internal:external employees does not seem to have a very profound impact. Having this cost driver could needlessly complicate the model without adding any value.
- **Project duration** - Project duration cannot be accurately predicted beforehand. The dataset contains information about the eventual duration of a project, and not
the initially planned duration. Similar to predicting costs, the duration depends on a number of other factors.

- **Old projects** - The training set still contains old (2009-2012) projects which may not be representative of the current day B/CAO organization.

Besides the raw data which is imperfect, we have to take into account that there is more to software development than the numbers which are fed into the neural network. To build a new product you need to have skilled and motivated developers to successfully go through the entire development process. Although employee satisfaction has been more or less stable, there is sufficient room for improvement as acknowledged by an extensive review by Effectory International [42] and the Belastingdienst management team. That report points out, amongst other issues, how B/CAO employees are unhappy with the efficiency of the entire organization and working conditions.

The current model does not take these kinds of parameters into account, nor does it consider how new and drastic changes in IT development methodologies could impact productivity. B/CAO is in the process of implementing alternative workplace strategies. While there have been pilot programs and some people already do work from home, such programs are scheduled to be expanded. Once the IT infrastructure and IT services are available to accommodate this new type of work environment, many people will want to switch. Such changes will also impact productivity but we do not yet know how.

Most predictions extrapolate from known sample data, that is not necessarily problematic but do keep in mind that parameter settings which are far beyond the normal range of expected values are less reliable than others. The confidence measure produced by the model and displayed in the tool which should deal with this, is actually not mathematically “correct” in the sense that it does not provide the exact upper and lower control limit at one standard deviation from the mean. So in practice you cannot definitively say that 68% of all real world projects will fall within the control limits as calculated by the predictive model. Instead it is a practical way to indicate the level of confidence a user should have in the predictions.

The model is also limited because it should be updated regularly to be in line with the most historical database of completed IT projects at B/CAO. But it does not update automatically, instead someone needs to “manually” update it as explained in appendix B.

Choosing to predict the number of hours rather than the hours per function point has turned out to be a mistake. Because the error function to evaluate the trained network is calculated out of the absolute error rather than relative errors, and the predicted values can be magnitudes apart, the model is biased towards fitting the larger efforts. Either the error function or the predictor variable should be adjusted to improve accuracy.
9. Conclusion

Analysis performed on the collected information points out how certain cost drivers have considerable effect on productivity. There are clear trends to be found in these cost drivers that impact the required investments to build and adapt IT systems.

These trends should be used to improve planning, and to alter productivity norms. Because the current stated productivity norm depends only on programming language, it is too rigid. Productivity norms must be decided in such a way that they are realistic and achievable. My advice is to fix this issue by implementing more fluid norms. This can be done by assuming a slightly lower value than the prediction of the neural network model. A lower value pushes towards improving, rather than accepting the current situation. If the model is found to be too opaque in its exact working, B/CAO might consider a non-linear norm per programming language based on size only. Such trendlines can be found in figure 7.1.

Better planning should result in reduced waste even before starting the project. If planning is tighter, yet still realistic, there is less opportunity for waste.

Conclusions with regards to the individual cost drivers are:

- Functional size and programming language are the most important cost drivers.
- Project duration has an effect contrary to what is assumed in the QSM model.
- Building an entirely new system is more costly than modifying existing software (for most programming languages).
- More expensive employees are not necessarily more productive.

9.1 Recommendations

I investigated the practical use of advanced learning algorithms in the context of cost estimation. While the volume of data in this research is limited, the techniques and ideas applied are especially suitable to big data analytics. Similarly we can apply the three crucial aspects of big data analytics: data, model and transformation [51] to this
Conclusion

research. Data is the use of information to give you a better view of what is happening within your organization. The analytical model is the mathematical foundation behind the logic that structures the data. And the essential but often overlooked part is the transformation where the company takes advantage of data-driven analysis. Because merely internalizing information without utilizing these newfound advantages does not lead to any improvements. Without acting on data, there is no value in it. The following recommendations are about improving data collection, the model, and the transformation.

9.1.1 Data

The current data is entirely based on objective measurements. Certain subjective assessments would offer more possibilities for any predictive model. To improve data collection, record more details for all new instances. Some of the cost drivers have been used in the predictive model but had to be manually retrieved from other sources. By storing this data in the very same location there is no need for the laborious and error-prone work of combining it. COCOMO II cost drivers are primarily subjective assessments about the properties of the software development process, by incorporating some of the same features in the B/CAO model we could certainly produce much better estimates. Issues such as complexity and specification quality have come up in evaluations. It is recommended to store the following information in Datamanager.

- FPA counting method (Nesma, IBRA, Cosmic)
- Project manager seniority (junior, medior, senior)
- Complexity (low, nominal, high)
- Quality of specifications (low, nominal, high)
- Proportion external project members
- % New function points

A project manager can easily provide estimates for each of these new properties.

9.1.2 Model

The way in which the cost model was generated uses a very standard machine learning algorithm. It is very likely that there are other methods or parameter settings which can produce marginally better predictive models. An expert could potentially reach better results using similar tools. If B/CAO is looking to improve the predictive performance of its cost model it should consider hiring experts or switching to commercially available machine learning solutions (in addition to storing the additional cost drivers as outlined in the previous section).
9.1.3 Transformation

Business analysis in itself is not a goal. The goal of business analysis is to make better decisions based on real data. But without acting on this data there is no value in any such analysis. Section 7.5 outlines the influences of cost drivers on software engineering, planning should be steered towards a better (more efficient) configuration of cost drivers.

Summarizing, my recommendations for B/CAO are to:

- Rethink the rigid official productivity norms.
- Formalize standards and decision making processes based on cost predictions.
- Instruct front line managers on how to use the cost estimation tool.
- Plan for an updated cost model.

9.2 Future Research

Designing, building and testing software is only part of B/CAO expenses. Most are related to indirect costs [41]. Mapping and investigating these costs seems to be even more worthwhile. One aspect which was not considered in the model is software quality. A sensible assumption is that a higher quality product requires a higher investment. Within the context of software a higher quality software product is assumed to require less maintenance, and allow for cheaper maintenance.

Product quality, maintenance costs, and indirect costs are all interesting aspects for further study.
Bibliography


[9] Lean is here to stay. 2010. URL http://bcao.belastingdienst.nl/bijlagen/groep/79dca79e45a0e49998248c0d8239f88b/overig/Brochure%20Lean%20IT.pdf.


A. Encog model

Appendix A describes the steps of preprocessing, generating a neural network using the Encog framework, and exporting the result to a javascript implementation. The result is a web-based javascript implementation so that it can interact with an easy to use graphical interface.

Data preprocessing

Input to the Encog machine learning framework is a .csv file containing data from all the completed projects. Those efforts that have been deemed unrepresentative of the typical work produced within the organization are left out as described in 6. The .csv file contains eleven columns, separated with a comma. The header line is:

*FP,CHANGE,TIME,PRESS,HOURLY,INTERN,NEWFP,L1P,L1,L2,HOURS.*

*HOURS* is the target variable while the others are cost drivers. Adding new projects to the dataset is as easy as entering a new line to the file with all the cost drivers and the target variable in the correct order. Normalization of these values happens within the Encog framework.

- *FP*: counted Nesma function points.
- *CHANGE*: proportional size in percentages of the programming effort compared to the overall system.
- *TIME*: turnaround time in calendar days.
- *PRESS*: *FP/TIME*.
- *HOURLY*: total costs divided by *HOURS*.
- *INTERN*: percentage of internal employees contribution.
- *INTERN*: percentage of new functionality.
- *L1P*: percentage of effort programmed in the primary programming language.
- *L1*: primary programming language.
  - *COBOL, JAVA, COGEN, C, JAVAI, IMW, SAS, SASB POWERBUILDER, IMW, OTHER*
- *L2*: the secondary programming language.
  - *NONE, COBOL, JAVA, OTHER*
- *HOURS*: the total number of hours spent.

Because the ensemble classifier actually comprises of 50 separate models, we have to create 50 of these. First, create 50 bootstrap samples out of the original data using the bootstrap.html script.
Appendix A. Encog model

(using the Google Chrome browser). The script “downloads” the 50 .csv files. Copy these and the original into a separate folder. Then open Encog.

Encog

After preparing a new folder which contains the .csv files formatted as described above, we will transform this data into a neural network. The properties of the neural network are such that it models the training data. Take the following steps to generate the neural networks and create a javascript file containing the model.

step 1 Click on File → ChangeDirectory/OpenProject → Select the correct folder.
step 2 Right click on the original .csv datafile → Analyst Wizard.
step 3 General → set the following settings:
  - Goal: Regression
  - TargetField: Hours
  - CSVFileHeaders: check
  - NormalizationRange: 0 to 1
  - MissingValues: MeanAndModeMissing

step 4 CodeGeneration → set the following settings:
  - GenerationLanguage: Javascript
  - EmbedData: check

step 5 Click Ok
step 6 In [filename].ega, under [DATA : STATS]. Change the line starting with “l1p”,1,... into “l1p”,0,... This tells Encog that l1p is not a class variable. Instead it is a continuous variable.
step 7 In [filename].ega, under [DATA : CLASSES]. Remove every line starting with “l1p”.
step 8 In [filename].ega, under [SEGREGATE : FILES]. Change “FILE_TRAIN”,75 and “FILE_EVAL”,25 into “FILE_TRAIN”,95 and “FILE_EVAL”,5 respectively. This way we train the model on a random subset.
step 9 task – full → Execute → Start
step 10 [filename].code.html now contains the neural network.

After performing these steps, [filename].ega contains this Encog script file which was used to produce the resulting output. Use the [filename].ega configuration in training the other neural networks. Simply perform step 9 for each of the generated bootstrap samples and copy the WEIGHTS variable in [filename].code.html into weights.js to include this model in the ensemble classifier.
B. Cost estimation model

The tool itself is a simple interface that is capable of visualizing the effects of cost drivers.

Figure B.1: A simple interface for the cost estimation tool allows the user to enter a number of cost drivers. The underlying model will compute a prediction and plot ranges of predictions for all the cost drivers.

The allowed ranges for each of the cost drivers are mentioned in the corresponding pop-over tooltip. It is not possible to leave one of the cost drivers blank. And only numeric inputs within the appropriate range are permitted, except for the language dropdown fields.

After filling in and selecting the appropriate cost drivers, press **Bereken** (calculate). B/CAO software cost estimation model will compute the exact prediction for that particular combination of cost drivers, and display how the productivity changes as a single cost driver is altered.

It is highly important to note that the graphs show the changing development of productivity as a single cost driver is altered, while all the others remain constant. This implies that e.g. the graph for productivity and function points, displays how productivity changes as the project grows or shrinks in size (in Nesma function points). **All the other cost drivers remain constant.**

The model will only calculate and show predictions for combinations of cost drivers that are considered to be possible. Cost drivers must be within the observed range for that programming language in the training set, otherwise that value is too high or too low. This will result in blank plots - no prediction is calculated and nothing is plotted. The end-user must still take care to correctly interpret the results,
It is possible to display several plot lines in the same graph, this way you can compare one project against another. But there are some limitations: if you initially plot the predictions for a 500 FP project which builds a 500 FP system, you cannot display the predictions for an 800 FP system in the same graph. By pressing Nieuw (new) you will clear all the plots, and allows you to start over.

Finally there is the option to change the graph sizes (small, medium and large).