FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Lean Forecasting In Software Projects

Pedro Manuel Costa Miranda

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Carlos Pascoal Faria

Second Supervisor: Filipe Figueiredo Correia

July 27, 2020

Lean Forecasting In Software Projects

Pedro Manuel Costa Miranda

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: António Miguel Pontes Pimenta Monteiro
External Examiner: Miguel António Sousa Abrunhosa Brito
Supervisor: João Carlos Pascoal Faria

July 27, 2020

Abstract

When developing a software project, it's recognisable that accurate estimations of development effort or duration play an important part in the successful management of the project. Although this process is so important, developers and experts can't usually estimate accurately the effort, time and cost of a project to be developed, due to the uncertainty that underlies their activity.

Even if the estimate is accurate enough so that delivery dates are respected, methods that rely on human estimation are, often, time consuming, what can represent a problem when teams waste precious time in making estimations.

In order to mitigate these problems, this dissertation aims to analyse and discuss the benefits of using data-driven methods to achieve a high degree of leanity while still achieving accurate forecasts. After analyse some of the existing methods for estimation and forecast, we chose the Monte Carlo method, a forecasting method that provides a set of positive points. We focus on this lean forecast method because of its automation capacity, that will help reduce the time teams waste on estimations, still delivering accurate results. This method is also easy to understand, implement and use, since the number of inputs required and the difficulty to collect these inputs are low. In order to better represent the problem, the output of the method contains a certain level of uncertainty represented by a confidence interval. In order to validate this method, a tool was implemented and some experiments were built in order to evaluate it in terms of error, when comparing to real values registered. Some comparisons with other methods were also made using for both this comparison and the evaluation a dataset provided by Fraunhofer. After this we conclude that the Monte Carlo method provides forecasts with a high degree of leanity and accuracy, verifying this way the accuracy and leanity of data-driven methods.

Keywords: Estimation, Forecasting, Software Development, Monte Carlo Simulation

Resumo

Quando se desenvolve um projeto de software, é reconhecível que estimativas precisas do esforço envolvido no desenvolvimento são uma parte importante na gestão bem-sucedida do projeto. Embora este processo seja tão importante, desenvolvedores e especialistas não conseguem normalmente estimar precisamente o esforço, tempo e custo que o projeto a ser desenvolvido terá. Isto é inerente à incerteza subjacente à sua atividade.

Mesmo que a estimativa seja precisa o suficiente para que as datas de entrega sejam respeitadas, métodos que dependem das estimativas de humanos consomem, normalmente, muito tempo, o que pode representar um problema quando equipas gastam tempo precioso a fazer estimativas.

De forma a mitigar estes problemas, esta dissertação analisa e discute os benefícios de usar métodos orientados por dados para atingir um nível alto de facilidade de utilização do método, enquanto se mantém a precisão da previsão. Depois de analisar alguns dos métodos existentes para estimação e previsão, foi escolhido o método de Monte Carlo, um método de previsão que proporciona um conjunto de pontos positivos. Escolhemos este método devido à capacidade de automação, que vai ajudar a reduzir os desperdícios de tempo da equipa em estimações, continuando a produzir previsões precisas. Este método é também fácil de entender, implementar e usar, visto que o número de dados de entrada e a sua dificuldade de recolha são reduzidas. De maneira a representar melhor o problema, os dados de saída do método contém algum nível de incerteza representada pelo uso de um intervalo de confiança. De maneira a validar este método, uma ferramenta foi implementada e alguns experimentos foram feitos de forma a avaliar as suas previsões em termos de erros, quando comparadas com os valores reais registados. Também foram feitas comparações com outros métodos, usando para estas comparações, bem como para a avaliação, um conjunto de dados fornecido pela Fraunhofer. Após isto, concluímos que o método de Monte Carlo proporciona previsões com níveis altos de facilidade de utilização e precisão, verificando assim a precisão e facilidade de utilização de métodos orientados a dados.

Keywords: Estimar, Prever, Desenvolvimento de Software, Simulação Monte Carlo

Acknowledgements

Firstly, I would like to thank the Faculty of Engineering of the University of Porto for all the support given during this dissertation and the past 5 years.

A special thanks to my Supervisor, João Pascoal Faria, and to my Co-Supervisor, Filipe Correia, for helping me and for always be available with helpful suggestions to resolve difficulties that raised during the dissertation. I would like to also thank Ahmed Fares for all the help and ideas given during the development of my dissertation.

I also want to thank Fraunhofer, represented by Ricardo Graça and Rui Castro, for the dataset provided and for the availability to answer questions about the information stored and its meaning.

Thank all my family, especially my mother and father, for always support my decisions and for helping me building my future.

To my friends and colleagues who accompanied me during these five years, and some of them from even before that, thanks for the friendship, moments and help during difficulties.

Pedro Manuel Costa Miranda

"Those who do not remember the past,
are condemned to repeat it."

George Santayana

Contents

List of Figures

List of Tables

Abbreviations

PROMESSA	PROject ManagEment intelligent aSSistAnt
COCOMO	Constructive Cost Model
SLIM	Software Lifecycle Management
ASD	Agile Software Development
XP	Extreme Programming
SEER-SEM	System Evaluation and Estimation of Resources – Software Estimation Model
ANN	Artificial Neural Networks
SVM	Support Vector Machine
BN	Bayesian Network
WIP	Work in Progress

Chapter 1

Introduction

Software developers can't usually estimate accurately the effort, time and cost of a project to be developed. This is inherit to the uncertainty that underlies their activity, since after the first estimation of the effort, the project may, with some likelihood, need to adapt to evolving circumstances, which may lead to changes in its scope, and consequently in delivery dates. These new circumstances may be impossible to predict in any estimation effort taken when the project started. These delays not only affect the development team but also other parts of the company, such as staffing or marketing. This could, in some situations, lead to the company losing time and in many situations the trust of stakeholders. A snowball effect that had its beginning in that poor estimation.

Methods that rely on human estimation are, often, time consuming, what can represent a problem when teams waste time in making estimations that will, often, underestimate the effort and duration. From this problem rises the lean forecasting techniques, which aim for accurate forecasts with less effort and time spent estimating for the team or developer.

1.1 Motivation

The motivations of this dissertation project are: the fact that effort predictions in the industry, most of the times, are done using expert-opinion methods that could be time consuming and subject to human bias [42], and when these type of methods are not used, reports are that other types of methods don't perform as well as they are described in the literature and have a certain degree of difficulty to be applied to agile context projects [23], being the expert-opinion solutions much more apt to this task. Most researchers, in the other hand, keep focusing on data-driven methods instead, believing this type of methods are the future due to their automation, increased efficacy and degree of leanity, and due to their capacity of providing good accuracy. The degree of difficulty to apply this data-driven methods is, this way, the greatest obstacle for the industry still not use this type of forecast.

1.2 Objectives

This dissertation is inserted in the context of the PROMESSA project, being developed in FEUP. The objectives of the dissertation are, this way, similar to the objectives of the project, namely:

- Identify the motivations and forces at play when making forecasts and what are the main reasons that could make a forecast fail in delivering a good accuracy.
- Determinate what methods and approaches may work better in achieving more accurate forecasts and that include the lean factor, that is, that are effortless for the user, in agile context.
- Implement a method that takes into account the findings.
- Put the methods and strategy to test using real-world data from different real projects and prove this way their efficacy and accuracy.
- Develop a plugin based on that implementation to integrate with a software project management in order to test the model accuracy and efficacy and provide it as a contribution.

1.3 Contributions

In terms of contributions, this dissertation helps in:

- Through the revision of the state of the art, highlight some of the existing methods for forecast or estimation and discuss some comparisons between those methods, drawing conclusions on which provides better accuracy and leanity.
- Providing a working implementation using different delivery forecasting approaches, that can be integrated with software development management tools.
- Through the evaluation of the model, show its positive and negative points, also validating the results obtained with other methods, achieving some conclusion on how to proceed when trying to achieve more lean and accurate forecasts.
- Setting the ground for future work and possible new implementation based on this one, through the conclusion achieved.

1.4 Hypothesis and Research Questions

This dissertation is based on a hypothesis that serves as a basis for its development and implementation. This hypothesis is:

"*Data-driven methods for software project forecasting can accurately be applied in agile projects, bringing benefits in terms of reduced forecasting errors and user's effort*"

This hypothesis is sub constructed in the following research questions:

- **RQ1:** Is the chosen method more efficient than the others?
- **RQ2:** Which type of data will this method work with as input?
- **RQ3:** What metric would be more useful for better forecasts?

1.5 Methodology

In this dissertation, after revising important notions and the state of the art, highlighting some of the existing methods for forecast and discuss some comparisons between those methods, arriving to some conclusions on which method is better suited to be used in order to provide better accuracy and leanity, we validated if this data-driven method proves itself and this type of methods to be lean, that is, requiring less effort for the developers and teams in terms of inputs needed and time cost, and still providing good accuracy, in comparison with expert-opinion methods. We validate it by building a model based on that method and three different delivery forecasting approaches and then putting this model and each of the approaches to evaluation, building some experiences that help validate it. We also test the model against other existing models.

1.6 Structure

The document has 4 more chapters. In Chapter 2, some background analysis is made of what earlier methods made the foundations of software development estimation and forecasting, providing a complete analysis of the most relevant methods found, structuring a more in-depth research about these methods, what benefits and limitation they present and a brief comparison between some of them. Is also goes through important movements and premises that this dissertation will take as bases for its development and what metrics of both measurement, for input and output, and validation will be mentioned and used during the dissertation.

In Chapter 3, three different delivery forecasting approaches are discussed, implemented, as well as the Monte Carlo method, and their outputs are described and examples of each shown.

In Chapter 4, several experiments for each delivery forecasting approach are implemented and their results are discussed and some conclusions about their effectiveness and accuracy are drawn. The validation of the hypothesis is also done and the research question who deconstructed it were answered.

In Chapter 5, the conclusions of this dissertation are drawn, some problems that arise during its development are mentioned and possible future work is discussed.

Chapter 2

Forecast and Estimate Methods

In this chapter we present an overview of concepts that provide the basis for the rest of the document. We will look into important notions for the better understanding of software development effort forecasting/estimation , look into methods that establish themselves as pioneers on this field, provide an overview of relevant breakthroughs and the overall state of the art in software development forecasting. We also take important conclusions for the next steps to take on this dissertation and look into important metrics both for input/output and validation that will be mentioned during the rest of the dissertation.

2.1 Mythical Man-Month

When talking about software development management and effort forecasting or estimation, we need to mention the Mythical Man-month phenomenon, elicited by Brooks [7]. In his book, F.Brooks mainly explains that adding more manpower during the development of the project will probably end up delaying even more the project. When bad predictions are made and deadlines are built on them and then the project starts to show signs of being delayed, a common response is to add more manpower to fulfill the deadline. This is a bad response, since the time to integrate the new developers into the project and the increased communication overhead will cause even more delays.

2.2 Estimates vs Forecasts

We will now look into the #NoEstimates movement [21], a movement that defends that maybe estimations are not the best way to predict, since often they are not accurate and are wrongly used to put pressure into teams to deliver. S.McConnell recommend some tips for making good predictions [28], one of them being "*Distinguish between estimates, targets, and commitments.*" reveling that these terms are many times compressed in the estimate definition, that in case of an

underrated estimation, will lead to teams not being able to correspond to the commitments and targets defined. That's why is important to look at estimates and see them for what they really are, and don't overuse them.

R.Jeffries also mentions that estimates are almost impossible to do in a way that they will be accurate when there's missing data or when requirements are vague, a characteristic they often present [21].

L.Keogh goes a bit further and presents another idea, a separation and definition of two different terms for effort prediction[25]. These definitions are:

- **Estimate**, a single value prediction of what will be the real effort.
- **Forecast**, a probabilistic prediction of a range of values or a exact value with confidence levels associated.

L.Keogh defends that forecasting, due to its probabilistic characteristics, will often provide more accurate predictions and more reliable ones that estimates.

2.3 Metrics

In software development effort forecasting and estimation, there are several metrics available to classify both inputs and outputs. Some of these metrics are better understandable, represent better the variable associated and are easier to obtained than others. In this section we will look into some of the most used metrics. We will also look into some of the most used validation metrics, metrics useful to test and compare different methods.

2.3.1 Takt Time

Takt is a German word for a rhythm or beat. Takt Time describes the regularity of that beat. In production line manufacturing, Takt Time is the rate at which each manufactured item is finished. Using Takt Time, a manufacturer is able to run the line at a pace that is in correspondence with demand. It also provides an easy way to work out how long it would take to produce large batches of items [12].

2.3.2 Story Points and Velocity

A story point is a metric used in agile project development management to estimate the difficulty of implementing a user story, which is an abstract measure of effort required to implement it. It's a number that tells the team about the difficulty level of the story, that could be related to complexities or risks. The story points can then be used to then calculate the velocity of the team. The velocity is the amount of stories a team completes in a determined time window, like a sprint.

2.3.3 Throughput

Throughput is the number of items completed in a given time window or iteration. It is a measure of output - how much is getting done. Measuring daily throughput as well as trends over time provides insight into both the rate and rhythm of delivery [6].

2.3.4 Cycle Time and WIP

Cycle time is the time that elapsed from the start of work on a given feature till it was finished. The Work in Progress (WIP) is the number of items that the team already started to work on but haven't finished them till the moment [6], that is, the amount of stories which cycle time hasn't end yet.

2.3.5 Error metrics

As validations metrics, the main used metrics are the Relative Error (RE) or Percentage Error (PE) with the Equation (2.1), the Magnitude of Relative Error(MRE) with the Equation (2.2), the Mean Magnitude of Relative Error (MMRE) with the Equation (2.3) and the Root Mean Square Relative Error with the Equation (2.4)

$$RE = \frac{\text{Actual value - Estimated value}}{\text{Actual value}} \tag{2.1}$$

$$MRE = |RE| \tag{2.2}$$

$$MMRE = \sum_{i=1}^{n} MRE_i \tag{2.3}$$

$$RMSRE = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} MRE_i^2} \tag{2.4}$$

These metrics will be used in the validation phase of the method implemented and developed and also to compare the state of the art methods.

2.3.6 Other metrics

Besides the metrics presented earlier, there are other highly used metrics. Lines of Code, number of features, number of interfaces, use cases are some examples of this other metrics used for inputs. For outputs we can have duration metrics like days, weeks, months or effort metrics like man-day or man-month.

2.4 Types of Methods

After this more in depth contextualization, we will start to analyse the exiting methods and models. The methods are divided into 4 main types [37]:

- **Empirical Methods**, methods that gather information by means of direct observation and experience.
- **Theory-Based Methods**, methods based on an explicit theory or logic model.
- **Regression Methods**, statistical methods for calculating the relationships between a dependent variable and one or many independent variables.
- **Machine Learning Methods**, methods that work with machine learning and focus on the learning ability inherent.

2.5 Empirical methods

In this section, we will look into some of the existing empirical methods, methods that gather information by means of direct observation and experience.

2.5.1 Expert-Opinion

Firstly, let's begin with the most used set of methods in the industry for agile context projects, expert-opinion methods, also known as expert-judgement methods.

The main advantage of this method is that the expert can analyse the differences between past project experiences and requirements of the new project to better estimate, also analysing the impact of new technologies, applications and programming languages.

Muhammad Usman et al. showed that this type of techniques are the most commonly used for estimate effort and duration of software projects when agile software development (ASD) methodologies like Scrum or Extreme Programming(XP) are being used [41].

Magne Jørgensen reported that 10 out of 16 studies reviewed showed that using expert-opinion based effort estimation methods led to more accurate effort estimates than using sophisticated formal forecasting models [22]. The main reasons appointed for this conclusions are:

- The large number of input variables that many methods require can lead to overfitting and lower accuracy when we're using small data sets to learn from.
- The effort and complexity of building a proper model.
- The difficulty of understanding what input variables are really important and those that are not.

This allied to the fact that expert-opinion methods are more easily explained and understood that other complex methods, shows that expert-opinion based methods are commonly better accepted by companies and their staff.

2.5.1.1 Wideband Delphi

Wideband delphi, developed by Barry Boehm and John Farquhar, is an estimation method by expert-opinion where the project manager selects the estimation team and a moderator [39].

The team should consist of 3 to 7 project team members. Also, the moderator should be familiar with the Delphi process, but should not have a stake in the outcome of the session if possible, to try to avoid human bias. If possible, the project manager should not be the moderator because he should ideally be part of the estimation team. Every member should understand the Delphi process, read the vision and scope document and any other documentation, and be familiar with the project background and needs. The team should agree on a unit of estimation and after some brainstorm, write down the assumptions made.

Individually, each member should right down his estimates and assumptions that lead him to that estimation. The estimations are collected and a plot is made with every estimate and the team resolves any issues or disagreements. However, no individual estimations are discussed and the team tries, based on that discussion, to arrive at a consensus. In the end the project manager and the team review the final results to check if everything is in order.

The main downsides of Wideband delphi are the great amount of time spent in estimation that could be used in other tasks and the subjectivity to human bias, because even though the method recommends that the moderator doesn't have a stake in the outcome of the meeting, this is not always ensured and other members in the meeting could make estimations to favor their own stakes.

2.5.1.2 Planning Poker

Planning poker, developed by J. Greening [14] and based on the Wideband delphi method, is an estimation method used during the release planning phase, when every story is getting a effort value and when the team makes the first draw of how the sprint will be organized. Basically, when using planning poker, a story is read and a brief explanation of the story is made so that every team member participating understands it. After that, every team member has to write down on a card their effort estimation for that particular story. Instead of writing down on a paper a value for every estimation, there are decks of cards available, like the classical deck in which the cards are numbered with the Fibonacci series in order to reflect the inherent uncertainty in estimating larger items [38]. Then, when every member has already his estimation made, the cards are shown. Based on the estimations of each member of the team, a discussion starts, to analyse the highest and lowest estimations and to try to understand their perspective and then try to achieve common ground among the team to proceed. If this common ground is not achieved, the team should just stop and defer the story, split it, or take the lowest estimation. Also if a story is getting consensus from the participants in attributing it a high value, the story should be split, since it probably represents a super story and can, probably, be divided in smaller ones.

In summary, this method aims to involve all the team members in the estimation process and make it more quick. Although being presented as a great technique, it presents some problems and issues that may rise:

- Teams could just ignore the discussion phase, a big feature of the method, and just take the most consensual value.
- Being a manual process and more time consuming than computational methods, what translate in more time spent in estimation.
- Subject to human bias, personal agendas of the participants who may influence the estimation in order to achieve personal goals.

Even though planning poker present some problems, is noted to be one of the most known and used methods in the industry. This could be explained by its simplicity and satisfactory accuracy. This method is revealed to be easy to understand, not only by members of the team, but also by stakeholders and upper management.

2.5.2 Analogy

Analogy methods are based in the analogical reasoning inherent to the human cognition. Analogical reasoning takes a fundamental part in recognition, classification, learning and in the process of creativity. Whenever we are about to make a decision, we use analogical reasoning by recalling related past experiences.

The use of analogy for software effort estimation is the process of forecast the effort and duration of a project, based on previous relevant analogous projects. Its main advantage is the fact that forecasts are based upon experiences that can be analysed to determine the specific similarities and their possible impacts on the new project.

According to Ali Idri et al. [17], analogy processes are, when automated, generally composed of three steps:

- **Characterizing:** Feature and project selection, feature weighting, and the selection of other dataset properties, such as dataset size, outliers, feature type, and missing values.
- **Similarity evaluation:** Retrieval of the cases that are the most similar to the project under development using similarity measures, in particular the Euclidean distance.
- **Adaptation:** Prediction of the effort of the target project based on the effort values of its closest analogs. This requires choosing the number of analogs and the adaptation strategy. The number of analogs refers to the number of similar projects to consider for generating the estimates. Based on the closest analogs, the effort of the new project is derived using an adaptation strategy.

The main advantages of Analogy based methods are:

- Can achieve good levels of accuracy.
- Simple if the organization repeats similar work.
- Estimates are immediately available.

Although presenting some advantages, when using this type of methods it is extremely tough to evaluate the dissimilarity among the environments and therefore assess the correctness of the data [38].

2.5.2.1 ANGEL

Chistopher Schofield's ANaloGy Estimation tooL (ANGEL) [34] is one of the most used analogy estimation tools. In this approach, for the characterization phase we must characterise a project with one or more descriptor features. These can be represented as quantifiable (interval, ratio or absolute) or categorical (nominal or ordinal) variables. These must be available at the point when an estimate of the goal feature (effort) is required. It is recommended that features that have a direct impact on effort are used. Collecting features that have no perceived relationship to the project effort is likely to result in the selection of poor analogies, what will lead to wrong forecasts. However, no expert is required to decide what features will be selected, since the ANGEL tool incorporates a mechanism for finding the best combination of features presented to it, based upon the historical data it is presented with.

For the similarity measurement phase, the measures that have found practical use are predominantly the nearest neighbour algorithms. The nearest neighbour algorithm used in forecasting by analogy involves measuring Euclidean Distance (2.5) in n dimensional space.

$$ED(x,y) = \sqrt{\sum_{i=1}^{i=n}(x_i - y_i)^2} \tag{2.5}$$

The Euclidean Distance represents the simplest proximity measure and the most commonly used. To use this measure, ANGEL first needs to standardise the features, dividing them by their range, so that every feature is one value between 0 and 1. Two different feature types can be incorporated into the similarity measure, both quantifiable and categorical features but while the use of quantifiable data is not problematic, categorical features have no notion of interval and therefore can only be described as identical, assuming the value of 1 or different, assuming the value of 0. For measuring the confidence that can be placed in a forecast, each project is successively removed and its effort estimated using the remaining projects as the analogy source. The estimated effort for each of the projects is then compared to their actual effort, which yields an indication of how much reliance can be placed upon new estimates from that project case base.

For the adaptation phase, ANGEL finds the n closest projects and makes an average of their total effort or alternatively applies a weighting so that the closest projects contribute more to the eventual estimate. The Angel tool has 4 different strategies: choosing 1 analogy (closest project), 2 analogies (either with the average or weighted average of the 2 closest projects) or 3 analogies

(average of the 3 closest projects), since more analogies would mean more computational power required. The main limitation of the ANGEL tool and analogy methods in general are:

- Not applicable in agile context since it takes final efforts of analogous projects and not the final efforts for each iteration of those projects.
- Is based on a partially wrong premise:"Similar projects have similar costs".

To dealing with noisy features, features that don't translate in useful analogies and only add noise, ANGEL performs an exhaustive comparison of every possible combination of features until the subset of features that returns the best confidence is found. This could be a problem since this exhaustive comparison can be computationally expensive. In 1998, Christopher Schofield estimated that in a set of 20 projects processed by a Pentium 200, ANGEL would take 5 seconds to process 5 features, 2 minutes and 40 seconds to process 10 and 45 hours to process 20 features [34]. This estimation shows the exponential behavior of this comparison.

The ANGEL tool was used to forecast effort in several databases to check which of the strategies (1, 2, 2 (weighted) or 3 analogies) was performing better. The Table 2.1 presents those results.

Data Sets	Nº Project Cases	Optimum Nº of analogies
Albrecht	24	2 (weighted)
Deshamais	77	3
Finnish	38	2 (weighted)
Hughes	33	1
Kemerer	15	2
Mermaid	28	1
RealTime1	21	2
Mermaid	18	1

Table 2.1: Optimum no. of analogies for each dataset, reported by Schofield [34]

Summarizing the table, 1 and 2 analogies work better than 3 and 2 (weighted) in most of the datasets analysed by Schofield.

2.6 Theory-Based Methods

In this section, we will look into some of the existing theory-based methods, methods based on an explicit theory or logic model.

2.6.1 COCOMO

COnstructive COst MOdel (COCOMO), a theory-based model, was created by Barry Boehm in 1981 and is one of the first probabilistic methods.

COCOMO is divided in Basic, Intermediate or Detailed type, depending on the processing power and detail required and these types use one of the following models:

- Organic, a mode used for small projects developed within stable environments with a relaxed requirement specification
- Embedded, a mode used in large projects that are operated under tight constraints.
- Semi-detached, a mode used in intermediate projects that lie between the two previous extreme models.

COCOMO takes as main inputs, all subject to scale factors, the LOC (Lines of Code) of the project and in the intermediate and detailed versions, it implements cost drivers. Cost drivers are refinements to the estimate to take account of local project characteristics. Cost drivers are spread across four categories (product, computer, personnel and project) and have ratings (Very Low, Low, Nominal, High, Very High) assigned, depending on the extent to which they affect the project in question.

The 17 available cost drivers are [29]:

- Product
  - Required system reliability
  - Complexity of system modules
  - Extent of documentation required
  - Size of database used
  - Required percentage of reusable components
- Computer
  - Execution time constraint
  - Volatility of development platform
  - Memory constraints
- Personnel
  - Capability of project analysts
  - Personnel continuity
  - Programmer capability
  - Programmer experience in project domain
  - Analyst experience in project domain
  - Language and tool experience
- Project
  - Use of software tools

- Development schedule compression
- Extent of multi-site working and quality of inter-site communications

The output effort is given in Man-month, with pessimist and optimist deviation forecasts.

2.6.2 Monte Carlo

Monte Carlo techniques are used to forecast the results of problems that can't be classified by a single exact value, or that are too difficult to solve in order to arrive to a single forecast value. Monte Carlo methods uses random sampling techniques, where the inputs are chosen randomly from a subset, and analysing the results over many computational cycles. Instead of a single value, a range of values is returned, forecasting the most probable result in that range of values. Random sampling techniques were first applied in the decade of 1930 to estimate the value of π, being later used, in 1946, by Stanislaw Ulam and Enrico Fermi to formalize the Monte Carlo method in the Los Alamos Laboratory, USA [27]. They used Monte Carlo methods to solve problems in the radiation shielding and neutron diffusion. The method only gained the name Monte Carlo after John von Neumann coined it, in memory of his uncle's favorite casino.

Nowadays, Monte Carlo methods are mainly used to manage risk in the financial field, oil and gas exploration and project management.

When applying the Monte Carlo method to software projects effort forecast, the simplest model would include the amount of work in the initial backlog, the columns where the work flow through (like Design, Develop, Test) and a cycle-time forecast for each of those columns. This model would allow simple forecasts, but it is not good enough to give a clear idea of the impact of events that occur during development that delay the goal date. Defects and bugs are found and added to the backlog of work, questions arise about the project and features and modifications in the project scope and design are made. Although these examples of variables are not essential to a project, when happening, they cause delays, many times doubling the duration of the project. So when implementing a Monte Carlo method we should pay attention to what variables are more important and which ones will impact the most the project's duration. This is the reason why a more granular model is so important.

In the end, the method returns a diagram representing the probabilities of ending the project by a given date, for example "*there is 70% chance that we will be done by the end of June, 80% chance it will be half of July*" [6].

We can use Monte Carlo method to forecast the defects costs, making two simulations, one taking in account defects defined previously and another one without these definitions. Subtracting the two simulations will give us the forecast for the amount of days that the team will spend fixing bugs and defects.

Another possibility is to forecast what staff skills are in demand, taking advantage of the description of the input columns. Columns are, usually, associated with specific staff skills. For example, the testing column is associated with testing skills, the development column is associated

with developing skills, etc. This way we can analyse what staff skills will impact the most the project duration and tackle this problem, arranging the team to achieve better forecasts.

Another advantage is that Monte Carlo methods can be integrated with Agile methods, working with iterations and, this way, improving the final forecast every sprint that has passed, since the model will before each sprint make readjustments in the final forecast with the information available from past sprint of the same project. Although it seems good, as stated by D.North in his blog [30], this can also lead to wrongest forecast, and so to make more accurate forecasts, some cautions must be taken:

- "*The past work should be representative of the future work*". If the past work was about adding new features and the next work will be about integrating the system with other systems, the past work is unlikely to be representative, since the team will have different paces in the task ahead.

- "*The future delivery context will not change significantly compared to the recent context*". If the team is changing or a reorganization has happened, this is likely to affect the delivery diagram and also change the parameters from that point on.

- "*The Monte Carlo function should be a reasonable indicator of the past work*". Identifying the parameters that genuinely capture the behaviour, and using them to define a function that reasonably represents the historical and future data, is hard.

In terms of practical experimentation of the method, J. Zang shows that when implementing a Monte Carlo model in a real project, the acceptance of business users, stakeholders and upper management was great [42]. Since the method returns a highly graphic diagram, as shown in Figure 2.1, it was easy for people with no knowledge in software development to better understand the premises of the forecast achieved. This shows another feature of Monte Carlo models. Since these models are easy to understand, in part due to its easy implementation, people outside of the area can deeply recognise the main factors playing during a software project development and more intrinsically understand the final forecast of the model.

There is an online tool implementing the Monte Carlo Model, developed by Sevawise[1]. This tools takes as inputs the historical data type (velocity or story count), the number of sprints per program increment(sum of the time windows of every sprints to analyse), historical data input (previous sprints in excel) and number of sprints of historical data(previous sprints). As output it returns a data grid which displays the level of confidence that a specific velocity or story count will be achieved for the program increment timebox selected. A useful feature implemented by Sevawise was the export to excel feature. This is highly advantageous because allows for better share of information between departments and better storage of past forecasts.

[1] More information about Sevawise tool can be found at https://sevawise.com/tools/monte-carlo

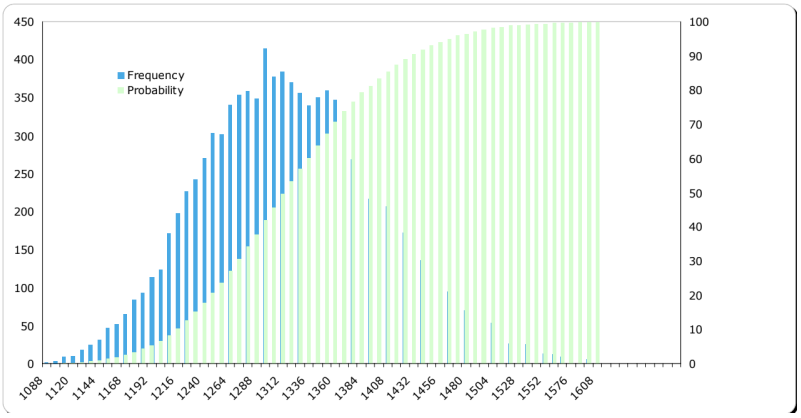

Figure 2.1: Example of a Monte Carlo diagram, represented by J. Zang [42]

Table 2.2 shows an example of an output grid from this model showing the percent chance some program increment velocity and sprint velocity will be achieved. Also classifies each forecasts for its bet, namely: Very conservative bet, Conservative bet, Realistic bet, Optimistic bet or Very Optimistic bet.

Percent Chance	Bet	PI Velocity	Sprint Velocity
99%	Very Conservative	93	31
80%	Conservative	124	41
50%	Realistic	142	47
20%	Optimistic	158	53
1%	Very Optimistic	181	60

Table 2.2: Example of a Output from Sevawise model, observed by Betts [4]

2.6.3 SLIM

The Software LIfecycle Management(SLIM) method, created by L. Putnam, is a theory-based technique to forecast software development effort.

This method is based on Putman's lifecycle analysis of the Rayleigh distribution. The Rayleigh distribution is used to forecast the number of developers that should be allocated to the development and maintenance during the lifecycle of a software project.

The SLIM model uses two main equations: the software productivity level equation and the manpower equation [11]. The software productivity level equation expresses the project size and development time, while the manpower equation expresses the buildup of manpower as a function of time.

This method was already very criticized by many researchers and Putnam himself found from implementing the method in 750 projects that the model worked properly for only 251. This method was also criticized by its underlying assumptions, including the fact that it disregards the initial stages (exploratory design /specification) of a project.

2.7 Regression Methods

In the present section, we will look deeper into a type of regression methods, methods that based themselves in statistical models for calculating the relationships between a dependent variable and one or many independent variables. One type of regression methods are the parametric models.

Parametric models are useful for subjecting uncertain situations to the rigors of a mathematical model. They usefully embody a great deal of prior experience and are less biased than human thought processes alone, what reduces the uncertainties of personal agendas and motivations of the forecast process. Parametric models provide forecast techniques based on facts, including mathematical equations as well as interpretation of historical data. Projects are break down into subcomponents, like stages for deliveries and match each stage with appropriate equation to obtain the forecasts. Although a main advantage of parametric models is the simplicity of implementation, unfortunately, this isn't always translated into simplification in terms of understanding of the methods by the team members or even the stakeholders and upper management.

2.7.1 SEER-SEM

The System Evaluation and Estimation of Resources-Software Estimation Model (SEER-SEM) is a parametric regression model developed by Galorath Inc., based on the Jensen model developed during the decade of 1970 by the Hughes Aircraft Company's Space and Communications Division.

SEER-SEM forecasts are based on knowledge bases, parametric algorithms and historical precedents. It makes forecasts of effort, duration, staffing, and defects.

SEER-SEM implements over 50 inputs that will influence directly the output of the method. Among them, 34 parameters are used by the SEER-SEM effort estimation model [20]. In SEER-SEM effort estimation, each input has sensitivity levels, with the ratings ranging from Very Low(negative) (VLo-) to Extra High(positive) (EHi+). Each main rating level is divided into three sub-ratings, such as VLo-, VLo, VLo+. These ratings are then translated to the corresponding quantitative value used to calculate the effort.

SEER-SEM takes as input the project scope, mainly size measurements (lines of code, function points, use cases), historic data, staffing levels and capabilities, requirements specification and volatility, development and target environment and economic factors. As outputs the SEER-SEM model has a variety of grids, diagrams and reports, as Figure 2.2 shows.

Initially, SEER-SEM generates a global effort and schedule forecast based on project size, complexity and productivity factors. Initial forecasts are generated in minutes. The problem with these simple forecasts is that they are not very accurate because they are based in little information about the project. To achieve better results the forecast must be refined increasing the number of inputs and the forecast complexity. Figure 2.3 shows the functioning of the SEER-SEM method through a software project effort forecast.

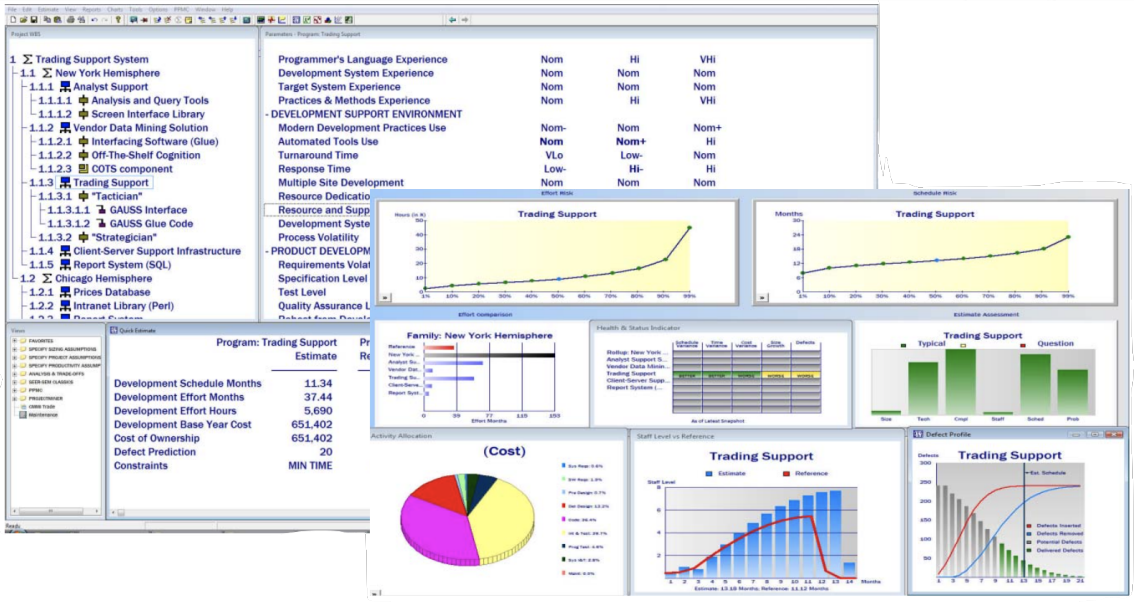

Figure 2.2: Example of the SEER-SEM output reports as presented in SEER-SEM product brief [20]

The main advantages of SEER-SEM is that produces an effort forecast with a good level of accuracy, the multi-platform integration ability, being able to be integrated with Microsoft Project, Microsoft Office, IBM Rational, and other 3rd-party applications [20] and its ability to be implemented in software projects using agile methodologies, what helps in maintaining the model apt nowadays, since Agile methodologies are the most used methods for software development management at the moment. Despite these two advantages, SEER-SEM present two main limitations that affect greatly its usage in the industry [38]:

- Over 50 input parameters related to the various factors of a project, which increases its complexity.
- The specific details of SEER-SEM make much harder to understand the relationship between the parameter inputs and the corresponding outputs.

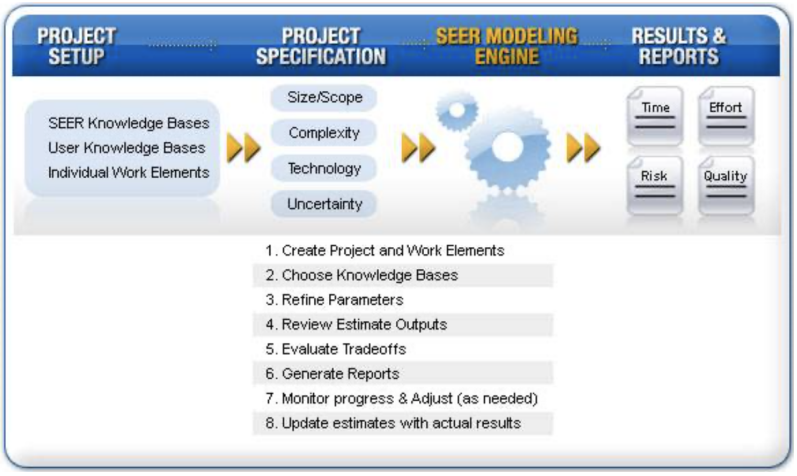

Figure 2.3: SEER-SEM functioning Diagram, reported in SEER-SEM product brief [20]

2.8 Machine Learning Methods

In this section, we will look into some of the existing machine learning methods, methods that work with learning types and focus on this machine learning ability.

2.8.1 Neural Network Model

Even if Neural Networks or Artificial Neural Networks (ANN) is regression based method, we will consider it as a Machine Learning method due to its learning particularities. Artificial Neural Networks are able to, in a more efficient and accurate way, establish complex relationships between the dependent (effort) and independent variables (inputs). It also has the ability to generalize from the training set of data. Most methods based on ANN use either feed-forward networks where no loops in the network path occur or feedback networks that have recursive loops. Limitations for ANN are:

- The difficulty of understanding why an ANN makes a particular decision [5], reason why this methods are many times called "*black boxes*", since it's really difficult to understand what is happening inside the model.
- Two forecasts from the same ANN will hardly be equal, due to the arbitrariness of the random weights.
- There are no guidelines for the construction of the neural networks (number of layers, number of units per layer, initial weights) [18]
- The amount of data required to usefully train a network since it's difficult to collect a considerable amount of data to train and test the model so it becomes viable.

Even with these limitations, there is a lot of interest in these models because of their ability of forecasting with good accuracy levels. S. Hydarali has conducted a survey where he asked

some questions about ANN to 33 developers [16]. Among other questions, when asked which method had better performance in software development effort forecasting, 60% of the developers replayed ANN, what translates this interest in ANN.

In the Figure 2.4 is represented an example of a simple neural network architecture with one input layer (having 3 neurons for each input), one hidden layer (with 3 neurons) and an one output layer.

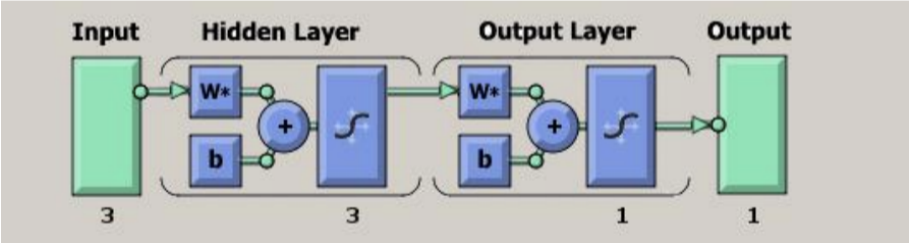

Figure 2.4: ANN Architecture example, explained by Bhatnagar [5]

A neuron computes a weighted sum of received inputs and generates an output if the sum exceeds a certain threshold. This output then acts as an excitatory or inhibitory input to other neurons in the network and the process continues until an output is generated. Artificial neurons are intended to be similar to biological ones, for example, the weighted inputs represent biological synapses, the interconnections between neurons represent the dendrites and axons, while the threshold function represents the activity in the biological neurocyton.

2.8.1.1 Support Vector Machine

Support Vector Machine (SVM) is a machine learning technique and classified by some researchers as a neural network method [9] used in various areas both for forecasting and classification. In a linearly separable case, the SVM classifier separates the instances from two different classes by using a hyper-plane that tries to maximize the margin, increasing the capacity of generalization of the classifier. The instances closer to the borders (or on the borders) are the support vectors. The number of support vectors represents the complexity of a model. In the Figure 2.5 is a representation of the SVM algorithm.

For applying SVMs in project forecasting, first, we must proceed to the calculation of the total number of story points and project velocity, that will be our inputs. This inputs are scaled using a nonlinear function, so that they are be within the range [0,1].

S.Satapathy et al. suggest the Function 2.6 [33], with X representing the values of the dataset, x the values of a project of the dataset and x' the scaled value.

$$x' = \frac{x - min(X)}{max(X) - min(X)} \tag{2.6}$$

After achieving the scaled inputs, the projects are separated in the hyper-plane in order of those scaled inputs. If the dataset is not linearly separable we can either use soft margin, where

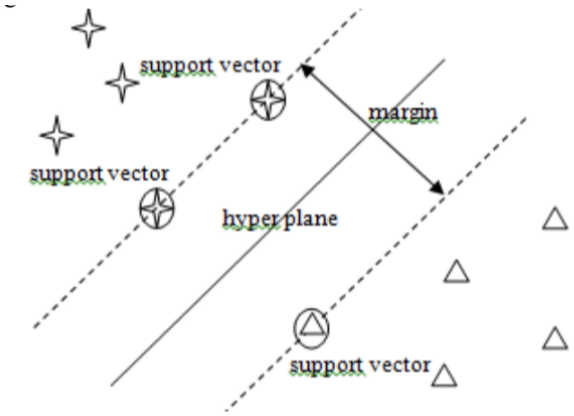

Figure 2.5: SVM Algorithm Representation, reported by Hidmi [15]

the separation is made as in linear mode but there's a tolerance for one or few misclassified dots, or use a kernel trick, that applies some transformations to the existing features and creates new features. These new features are the key for SVM to find the nonlinear decision boundary. Kernel trick uses one of the following non-linear kernel methods:

- **SVM Polynomial Kernel**, a classifier that generates new values by applying the polynomial combination of all the existing ones.
- **SVM RBF Kernel**, a classifier that generates new values by measuring the distance between all other dots to a specific dot.

S.Satapathy et al. compared the different kernel methods accuracy when applying them to real project datasets [33]. The Table 2.3 illustrates the results obtained.

Actual Effort	SVM Linear Effort	SVM Polynomial Effort	SVM RBF Effort
0.4615	0.3436	0.2332	0.4194
0.7692	0.7698	1.1027	0.7865
0.2637	0.2295	0.2059	0.2325
1.0000	0.7030	1.9282	0.8908
0.1538	0.1635	0.1691	0.1489

Table 2.3: Comparison between different kernel methods, reported by Satapathy et al. [33]

This results show that the RBF kernel method achieves a better accuracy than the rest of the methods.

E. Kocaguneli et al. implemented a SVM model and used it to forecast the effort of various datasets of software projects available in the database PROMISE[2] [26]. They concluded that the SVM model was the one that achieved better results among the models they tested.

[2]More information about the Promise dataset in http://promise.site.uottawa.ca/SERepository/datasets-page.html

Finally, this method can also be applied in forecasts of software projects effort when using agile methodologies and since has a increased capability of generalization, it's not confined to a specific type of project, which is very helpful for companies that work with different types of projects, like consulting companies. The main problem with SVM is the difficulty to implement and for non-experts to understand it.

2.8.2 Bayesian Network Model

Bayesian networks (BN) are graphical models that describe probabilistic relationships between related variables. BNs have been used for decision-making in problems that involve uncertainty and probabilistic reasoning. The first applications of BNs focused on classical diagnostic problems in medicine (during the 80s and 90s) [1]. The medical and biological decision-support domain was and continues to be the area in which more BN applications are published. However, companies like Microsoft have also used BNs for fault diagnosis and software companies have started to look into this method's benefits.

A BN is represented by a pair BN = (G,P), where G is a directed acyclic graph, and P is a set of local probability distributions for all the variables in the network. The directed acyclic graph ($G = [V(G), E(G)]$) is composed of a finite, non empty set of tuples $V(G) = \{(s_n, V_n), (s_{n+1}, V_{n+1}), ...\}$ and a finite set of directed edges $E(G) \subseteq V(G) \times V(G)$.

Each node V takes on a certain set of variables s. Each node also contanins the associated probability table, called the Node Probability Table, with the probability distribution of the variables of s.

The edges $E(G) = \{i, j\}$ represent dependencies among variables. A directed edge (i, j) from V_i to V_j shows that $V_i(parentnode)$ is a direct cause of $V_j(childnode)$.

If V has no parents, its probability distribution is unconditional. The probability distribution of variables must satisfy the Markov condition, which states that each variable of a node V is independent of non-descendants Nodes [8].

The 4 main advantages of Bayesian Networks are mentioned by Roger [31]:

- capability of handling missing data.
- capability of learning causal relationships.
- capability of combining prior knowledge and data.
- capability of avoiding overfitting the data.

The method takes as inputs mainly the project scope (user stories, features or lines of code), but can also implement other metrics in order to improve its accuracy. This other metrics could be staff factors (developer skills, experience and motivation) or process characteristics(organization and maturity of the development process).

The output of this method is a single point forecast with probability bounds, providing some essential uncertainty to the forecast value.

M. Perkusich et al. made a survey and an analysis of the applicability of Bayesian Networks in agile context software projects, more specifically, projects using scrum as its software development management methodology [32]. With the results of the survey and the opinion of some experts, they build the model, which directed acyclic graph for the key elements identified by them is partially represented in Figure 2.6.

Figure 2.6: Partial directed acyclic graph for the BN, defined by M. Perkusich et al. [32]

2.9 Fuzzy Logic

Fuzzy logic was founded and first applied by Lotfi Zadeh in 1965. Fuzzy logic is a mathematical tool for dealing with uncertainty and also it provides a technique to deal with imprecision and information granularity. Fuzzy reasoning is divided in 3 main components [3]:

- Fuzzification process, where the objective term is transformed into a fuzzy concept. Membership functions are applied to the actual values of variables to determine the confidence factor. This allows for inputs and outputs to be expressed in linguistic terms.
- Inference, a defuzzification of the conditions of the rules and propagation of the confidence factors of the conditions to the conclusion of the rules. A number of rules will be fired and the inference engine assigned the particular outcome with the maximum membership value from all the fired rules.
- Defuzzification process, where the translation of fuzzy output into objectives is made.

A membership function, used in the fuzzification process, is a curve that defines how each point in the input space (also referred to as the universe of discourse) is mapped to a degree of membership between 0 and 1 [2]. The main types of membership functions used are the Triangular Function (TMF), Trapezoidal Function and the Gaussian Function (GMF).

D. Ahlawat and R. Chawla tested these 3 meambership functions to observe which one presented better results [2]. Table 2.4 shows the results obtained.

Actual Effort	E.E with Trap	E.E with TMF	E.E with GMF
2040	2089	2075	1945
321	201.6	200.4	187.8
79	102	101.15	94.35
6600	7373	7329.2	6854.7
724	690.9	687.14	643.9
83	95.71	95.14	89.2
156	159.12	158.04	148.32
15	16.3	16.22	15.17

Table 2.4: Comparison between different Membership Functions, reported by D. Ahlawat and R. Chawla [2]

As we can conclude, the Gaussian membership function is the one that achieves better accuracy, according to this study.

During the inference process, fuzzy sets and fuzzy operators are used. Fuzzy sets are the subjects while fuzzy operators are the verbs of fuzzy logic. Fuzzy logic is comprised by conditional statements formulated as the rule statements "*if-then*". A single fuzzy if-then rule takes the form *if x is A then y is B* where A and B are linguistic values defined by fuzzy sets on the ranges of X and Y. The if-part of the rule "*x is A*" is called the premise or antecedent, while the then-part "*y is B*" is called the conclusion or consequent [2].

One major problem of software development effort forecasting is that data available in the initial stages of a project is often unclear, inconsistent, incomplete and uncertain. A fuzzy model is more appropriate when the systems are not suitable for analysis by conventional approaches or when the available data is uncertain, inaccurate or vague. Some of the difficulties of other methods, as described by Gray and MacDonell [13], are :

- Providing exact values for inputs, with methods often using values that they anticipate will occur, since for many metrics the actual value is never known with certainty until the project is completed.

- Over commitment, with most of methods committing to a value and not even providing confidence intervals.

- The size of data sets, since most of the times the datasets used to forecast the effort of the project are too small, not contributing with significant values to generate a good forecast.

- Interpretability of models, with some models being too hard to understand or giving outputs that are too hard to interpret.

These are problems that fuzzy models focus on solving, with easy interpretation of the model and output, handle of missing data, handle of non exact linguistic values and reducing the commitment through fuzzy outputs.

Based on these benefits of Fuzzy Logic, implementations of Fuzzy Systems combined with other existing methods have also appear.

V.Sharma and H.Verma proposed a Fuzzy system implemented on the Cocomo model, using the Gaussian Membership Function, since on their study they also reported the best accuracy of this membership function [36]. After establishing the fuzzy rules for the nominal effort components of Cocomo, they also proceeded to the fuzzification of the cost drivers available in Cocomo and subsequent establishment of fuzzy rules for these components.

W.Du et al. developed a fuzzy system implemented with the SEER-SEM model [10]. With this combination of processes, they helped to mitigate some of the problems and limitations of SEER-SEM model. After the development of the model, they putted it to test against the SEER-SEM model alone and were able to achieve good conclusions. In summary, the combination of methods proved in every experiment to be more accurate than SEER-SEM alone, also mitigating some of its limitations, already described.

A.Idri et al. proposed a fuzzy analogy system, a combination of the fuzzy logic method and analogy method [19]. Like a normal analogy method, the identification of cases, retrieval of similar cases and cases adaptation steps are still present in this model, but here these are fuzzification of its equivalent in the classical analogy procedure. With the implementation of fuzzy logic, they were able to introduce some machine learning abilities like the three criteria part of human nature that machine learning systems implement: tolerance of imprecision, learning and uncertainty. This method also help updating the false premise of analogy methods "*similar projects have similar costs.*" to a premise with some uncertanty added "*'similar projects have possibly similar costs'.*"

2.10 Final Discussion

After a more in depth analysis of each method, this section will summarize the methods seen and compare their benefits, limitations and accuracy. In this section we also highlight some researches who made comparisions between methods.

COCOMO, in synthesis, takes too much inputs when using the intermediate or Detailed types, because when using the Basic type, the accuracy is lower. Most of these inputs are expert-based, meaning that a expert must estimate some inputs in order to make the forecast. This could lead to subjectivity to human bias. Some benefits are the fact that it is simple to implement and is also relatively simple to understand. A possible implementation that mitigates some of the problems of COCOMO and increases its accuracy is combining it with fuzzy logic. The isn't easily applied in agile contexts, since it doesn't analyse past iterations of a project. In terms of accuracy, based on the findings, we can say that the results produced are not very accurate.

SLIM's major limitation is the fact that the method in a experience of 750 projects only worked properly in 251, what represent a high number of projects where the model didn't make any sence.

Also it is a target of criticism because of its assumptions, like the disregard of initial stages of a project. Compared to the other methods presented, SLIM doesn't add anything that could be considered as a benefit in relation to the others. The method is difficult to apply in agile contexts since it doesn't analyse past sprints of a given project. However, it provides a satisfactory accuracy and is relatively easy to understand.

Both Wideband Delphi and Planning Poker have proven to achieve a satisfactory accuracy and because of their simplicity, don't increase the difficulty of understanding the results. Their main liabilities are the fact that more time is necessary to make the meetings for making the estimation and is also subject to human bias. A great benefit both Planning Poker and Wideband delphi have, in relation to the others, is the fact that development teams are more comfortable using this expert-opinion methods because of the discussion between different perspectives and the great communication it provides. Among the expert-opinion methods, planning poker is simpler and with a more focus discussion achieves better results than wideband delphi. Both methods can be used in agile context, since new meetings can be done for estimating at the end of every iteration.

Angel, for being a analogy method, proves itself to be easy to understand and to interpret. The main limitations are the inability to handle missing data and non exact values and the fact that the premise of this type of methods is partly wrong. These limitations make it produce not very accurate estimations. When combining methods like Angel with Fuzzy logic, we saw that these limitations are mitigated and better accuracy is achieved. The method is not easily implemented in agile contexts since it doesn't analyse past iterations of the same project.

Monte-Carlo proves itself to be a method that presents good accuracy and besides that, is easy to implement and easy to understand, with some literature even giving simple examples of the process of this method. Another great benefit is the granularity inherent, with a easy interpretation of the degree each input affects the final forecast. The method is easily implemented in agile contexts. Based on the findings, we can say that the method produces accurate forecasts.

SEER-SEM, although showing satisfactory accuracy, presents itself as a hard method to understand and interpret, partially because of its numerous inputs (could go over the 50 inputs) and because of its specific details that make really hard the interpretation of the relation between inputs and outputs. When used in combination with Fuzzy logic, this difficult understanding is mitigated and the accuracy even improves, in relation to SEER-SEM alone. The method can be adopted when using agile methodologies.

Both SVM and BN are presented as difficult to understand, with some studies even calling these methods, black boxes. However, these methods show great accuracy, the ability to handle missing data, the fact of avoiding overfitting to data and the capability of generalization, not being confined to one type of projects, what translates to these methods producing accurate results. Both methods can work in agile contexts, since both can analyse past iterations of a project.

Fuzzy Logic also handles missing data, avoids overfitting and has satisfactory accuracy but, unlike SVM and BN, since it works with linguistic values, is easier to understand and interpret, what presents a great benefit. This method can also be applied in agile context projects since it can be implemented to analyse past iterations of the project.

Table 2.5 synthesises the methods limitations, benefits and their applicability in agile context projects, based on the findings of the literature revised in this chapter.

Method	Understandability	Accuracy	Overfits to Data	Agile
Cocomo	Easy	Not Good	Yes	No
SLIM	Easy	Satisfactory	Yes	No
Wideband Delphi	Easy	Satisfactory	No	Yes
Planning Poker	Easy	Satisfactory	No	Yes
Angel	Easy	Not Good	Yes	No
Monte-Carlo	Easy	Good	Yes	Yes
SEER-SEM	Hard	Satisfactory	Yes	Yes
SVM	Hard	Good	No	Yes
BN	Hard	Good	No	Yes
Fuzzy Logic	Easy	Satisfactory	No	Yes

Table 2.5: Synthesis of methods

D. Toka and O.Turetken compared the SEER-SEM method, the SLIM method and the COCOMO method when applied to a project effort forecast [40]. Table 2.6 shows the results obtained.

Method	MMRE	PE
COCOMO	74%	-54%
SEER-SEM	36%	-10%
SLIM	41%	-7%

Table 2.6: Comparison of different methods, reported by Toka and Turetken [40]

This results show that COCOMO provides the least accurate results.

B. Boehm defends [23] that the reason why expert-opinion methods are being more used in the industry and the reason why they show better results in the industry and then don't correspond in the literature, when compared to other types of methods is because these other types are many times wrongly used in companies. Companies often don't make good implementations of the methods to their specificity cases. This will lead to methods not corresponding to what researchers claim in the literature.

M. Jørgensen defends [24] that most times methods that work with ranges instead of exact values, have the ranges too narrow. Both the minimum and maximum values of the range are too close what very often doesn't translate the uncertainty inherent to the project, making the forecast more probable to be inaccurate.

In conclusion, the most promising methods are the machine learning methods and the Monte Carlo method due to their multiple advantages, described above. In the next chapters we opt to implement the Monte Carlo method for validate the hypothesis, due to its greater ease to be implemented and understood.

Chapter 3

Implementation

In this chapter we analyse and discuss the various delivery forecasting approaches followed under the Monte-Carlo method in this dissertation, as well as the metrics used in these approaches. We briefly describe the dataset provided by Fraunhofer. After that, we describe and discuss the implementations of the Monte-Carlo method core algorithm and each one of the delivery forecasting approaches mentioned and analysed in this chapter.

3.1 Goals

To help proving that data-driven methods can help in providing lean and accurate forecasts, we chose the Monte-Carlo method, as there are already some people using it [12][42], but there's still a lack of clear evidences of its efficacy and under which conditions it's most effective. The Monte-Carlo method main benefits are: the applicability in agile context, its output, being easy to understand and implement, its granularity and the good accuracy it provides. One of the goals is to, using the Monte Carlo Method, explore which forecasting approaches provides the better results. With this data-driven method and approach selected, we set to achieve the goal of proving the hypothesis of this dissertation, building a method that obtains a smaller error and greater accuracy than expert-opinion methods, always with condition of keeping the method simple and as lean as possible.

3.2 Dataset Schema Assumptions

In this section we describe the structure of the dataset available during this project, so that the algorithms shown next are easier to understand. The dataset was extracted from Jira, so the approaches were developed with the jira structure in mind, what means that they have the potential to be applied in other contexts where Jira is used. This dataset was store on a PostgreSQL database.

The dataset contains several projects, of which we gave preference to the projects developed in the later years (2018, 2019, 2020), since those are better representative of the present days. Each project has an *id*, *name* and *type* of project. The projects are divided in sprints. Each sprint has an *id*, *state* referring to the status of the sprint, *start date*, *end date* and the *goal*, that has additional information about the sprint. Each project also has its many stories. Each story has an *id*, *summary* of the story, a *type* of story, the *created* date, the developer *assigned*, the *creator* of the story, the *reporter*, an *project* id referring to the project where the story is inserted, a *status* of the story, the *priority*, a list of *sprints* where the story was developed, the *resolution date*, a *effort estimate* for the story(some stories don't have information on this field) and a list of *change log* items. The change log items work as a registration of changes that were made to a story. Each change log item has a *id*, a *summary* reveling which part of the story was updated, the *author* of the change, the *From* field representing the value originally, the *To* field, representing the value updated and the *date* of this change. The users who create stories, report, are assigned to stories and make changes to stories are registered having information about their *id* and *name*. Figure 3.1 represents the diagram of the dataset structure.

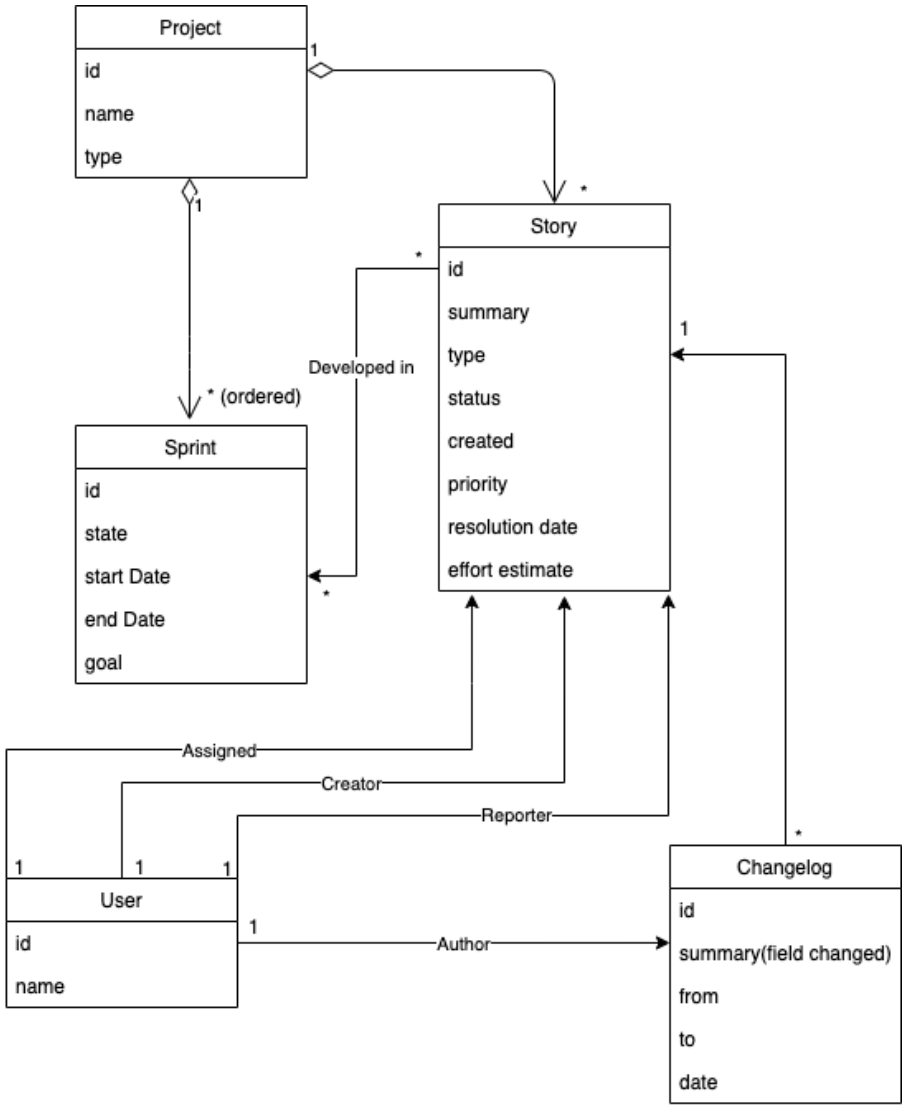

Figure 3.1: Dataset Structure Diagram

3.3 Delivery Forecasting Approaches

First we will introduce the 3 delivery forecasting approaches implemented, based on duration, takt time and effort. **Duration** measures the time between the beginning and end of a story development. **Takt time** measures the times between completion dates of consecutive stories. **Effort** measures the amount of time spent on developing a story. All these provide different ways to achieve a delivery date forecast. These 3 approaches were selected since these are all very common metrics in the field and after the state of the art analysis, our confidence was that these approaches could deliver good results in terms of accuracy and leanity.

3.3.1 Duration

When forecasting based on duration, the goal is to forecast how long will a story or a set of stories take to be completed and ready to be delivered. In this approach, we take as input N stories as historical data to output the forecast of M stories in the future. The duration d_i, with $i = 1,2,3,...,N$, of past stories is calculated as the difference between the date the story was put in the state *"In Progress"* (s_i) and the date the story was put in the state *"Done"* (f_i). We do this instead of calculating d_i as the difference between the beginning and end of a story to achieve more accurate results. The final forecast outputted D is calculated as the sum of the M forecasts, that can be used as a delivery forecast if the user stories are implemented sequentially.

3.3.2 Takt Time

Takt time is an another approach when trying to forecast a delivery date. When using this measurement of throughput, the date of the final delivery D outputted is calculated through the sum of the M forecasts, using as input the N takt times t_i, with $i = 1,2,3,...,N$, calculated through the difference between the completion dates of the historical stories c_n, with $n = 1,2,3,...,N+1$. This way we extract the amount of time between the finishing dates of a set of tasks, that is, the amount of time no story was completed, achieving development rates. With takt times, we can forecast how long will a story or a set of stories take to be completed.

3.3.3 Effort

Effort is another way to go when trying to make forecast for a story or a set of stories. While the previous two approaches used simply the time to forecast how long would it take to complete some stories, with effort we can do something else besides that, since we can also bring into the forecast the discussion about working rates of a certain developer. This method works by calculating the amount of time the developer assigned to the story was really working on that story, analysing the amount of other stories that were also assigned to him at the time that specific story was put in the state *"In Progress"* till the time the story was put in the state *"Done"*. In short this approach analyses the time a developer spent really working on a specific story. We assume that, when a developer is assigned to multiple stories, he splits his time equally to those stories. Due to lack of further information, we didn't take into account non-working days(holidays, etc.)

This approach takes as input the amount of time the developer assigned to the story was really working on that story, counting the amount of other stories that were also assigned to him at the time that specific story was put in the state *"In Progress"* till the time the story was put in the state *"Done"*, that is, for a set of N historical stories, the effort of each story fe_j, with $j = 1,2,3,...,N$ is calculated as the sum of eph_i, with $i = 1,2,3,...,h_j$, and h_j being the number of hours between the date the story was put in the state *"In Progress"* s_i and the date the story was put in the state *"Done"* f_i. At each hour i of this duration, the number of stories st_j in the state *"In Progress"* and assigned to the developer responsible for the story u_j is calculated and the effort eph_i is calculated

dividing st_j per 1, representing the effort per hour spent on a story. In short this approach analyses the time a developer spent really working on a specific story.

The approach outputs a W working effort for a number M of stories, taking into account the N historical efforts. Based on such forecast, and knowledge about the availability of the developers(full time, part time, etc.) for the project, the users may easily arrive at a delivery date forecast.

3.4 Forecasting Algorithms

In this section the algorithms created to implement the Monte Carlo method and the approaches are presented and explained. The language used is Python3, due to its ease to write and read, its improved productivity, being dynamically typed and being free and open-source.

3.4.1 Monte Carlo method

In order to implement the core algorithm of the Monte Carlo method we defined a function that receives the historical data and the number of tasks/stories to forecast and returns the median, mean, optimist and pessimist forecast, that work as the confidence interval of the forecast.

The Monte Carlo method was then implemented as follows.

```
1  def monteCarlo(historical, tasks, confidence):
2     num_reps = 200    # number of repetitions
3     target = []
4     r = 0
5     while r < num_reps:
6        b = 0
7        obj = 0
8        while b < tasks:
9           obj = obj + np.random.choice(historical)
10          b = b + 1
11       target.append(obj)
12       r = r + 1
13    target.sort()
14    rem = int(confidence * len(target))
15    if rem > 0:
16       target = target[rem:-rem]
17    mean = statistics.mean(target)
18    median = statistics.median(target)
19    lower = min(target)
20    upper = max(target)
21
22    return [median, mean, lower, upper]
```

Given a set of historical values (for effort, duration or takt time), the algorithm selects a random value from that historical data, adding them together to get the value for all the stories. We then have a set of values for the specific number of stories and after sorting this set, we exclude a percentage of confidence on each side, getting this way a certain confidence interval. We then return the mean, median, optimist (lower) and pessimist (upper) forecasts for that number of stories. The median is classified as our forecast, being our confidence interval defined by the optimist and pessimist values.

In our implementation we use 200 repetitions, due the results obtained. For all the projects in the dataset, with the takt time approach, using half of the project as historical data points(stories) and the other half to validate the forecasts, we analyse the values of MMRE produced with different numbers of repetitions, in relation to the processing time it took to calculate these MMRE. Table 3.1 presents these results. We assume that the other approaches would lead to similar results and conclusions.

Number of Repetitions	MMRE	Processing Time
100	32.4%	4 seconds
150	32.3%	6 seconds
200	31.4%	8 seconds
250	32.2%	10 seconds
300	32.3%	12 seconds

Table 3.1: Ideal Number of Repetitions

In all the following experiences, the confidence parameter was set to 2.5%, getting this way a 95% confidence interval on the forecast.

3.4.2 Duration

When using the duration approach, we select the date a story was put in the state *"In Progress"* and the date the same story was put in the state *"Done"*, this way outlining the time a story was being worked on. We use the time difference between those dates as historical data and then run the Monte Carlo method with it and get the forecast.

The code snippet of the first part of this approach is as represented below.

```
1 def duration(idProject):
2     cursor.execute("SELECT a.date, c.date FROM changelog a INNER JOIN output b ON a
          .id = ANY(b.changelog) AND a.toString = 'In Progress' INNER JOIN changelog
          c ON c.id = ANY(b.changelog) AND c.toString = 'Done' AND b.project=" + str(
          idProject))
3     extract = cursor.fetchall()
4     start = [i[0] for i in extract]
5     finish = [n[1] for n in extract]
```

```
6     diff = [x2 - x1 for (x1, x2) in zip(start, finish)]
7     historical = []
8     minLimit = 300
9     maxLimit = 4320000
10    for x in enumerate(diff):
11        if(x.total_seconds() > minLimit and x.total_seconds() < maxLimit):
12            historical.append(x.total_seconds())
13    return historical
```

We extract from the dataset the values of the moment the story was put in the state *"In Progress"* and then the moment the story was put in the state *"Done"*. The duration is calculated for each element and then values outside the limits are removed.

Limits were defined after the analysis of the first and third quartiles of the data extracted in order to define outliers and remove unrealistic values. After creating a boxplot with the duration of all the stories of all projects available in the Fraunhofer dataset, we identified that values smaller than 300 seconds (5 minutes) and bigger than 4320000 (50 days), respectively the first (25%) and third (75%) quartile of the entire dataset, should be considered unrealistic, that is, values that don't correspond to realistic situations. With this in mind, we only select for historical data, values that are within the range defined by this two values. The Figure 3.2 represent the boxplot, referred above, in days for better perception.

Since every project has its own particularities, when selecting the start and finish dates of the stories, we restrict the historical data to only include data about past stories of the specific project we are forecasting for. This way we exclude possible errors that could raise from picking data about stories that represent projects with very different goals.

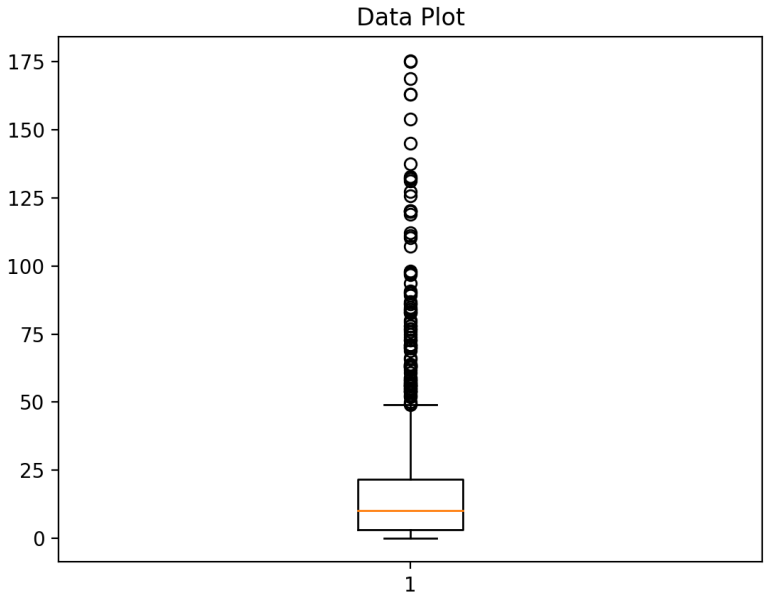

Figure 3.2: Boxplot of the time for completion of a story in days in the dataset used

3.4.3 Takt Time

In the Takt Time approach, we select the completion date of each of the stories. With the completion dates ordered, we calculate the difference between one date and the next, getting this way the time between the completion of two stories. Using these takt times as historical data, we run the Monte Carlo method to obtain the forecast. Below is represented a code snippet for extracting the relevant historical data from the database.

```
1  def taktTime(idProject):
2
3     cursor.execute("SELECT resolutionDate FROM output WHERE resolutionDate IS NOT
          NULL AND project =" + str(idProject))
4     resolution = cursor.fetchall()
5     resolution= [i[0] for i in resolution]
6     resolution.sort()
7     historical = []
8     maxLimit = 259200
9     l = 0
10    while l < len(resolution)-1:
11       timeInt = resolution[l + 1] - resolution[l]
12       if(timeInt.total_seconds() < maxLimit):
13             historical.append(timeInt.total_seconds())
14       l += 1
```

```
15
16    return historical
```

We extract from the dataset the resolution dates of the stories of the project. After sorting them, to make sure they are ordered by date, we calculate the takt times between the dates and check if these takt time are within the limits defined.

A boxplot was generated with all the takt times of the Fraunhofer dataset, allowing for the identification of values bigger than 259200 seconds (3 days) as outliers of the dataset. This way, as done in the duration approach, the range of acceptable values is defined as values below the dataset third quartile. This boxplot is represented in Figure 3.3. Due to the great amount of outliers above 3 days, the boxplot was zoomed to allow a better perception of where the third quartile ends. The median is the golden line, being set really close to zero. This happens due to the fact that a lot of small values are present in the dataset. The removal of values outside the dataset is done in this approach to ensure that breaks on productivity don't affect the accuracy of the model.

As explained above, since every project has its own particularities, and since many projects are developed at the same time in a company, what would lead to the comparison of stories of different projects, leading to wrong values, we only select the stories of the project we are forecasting for.

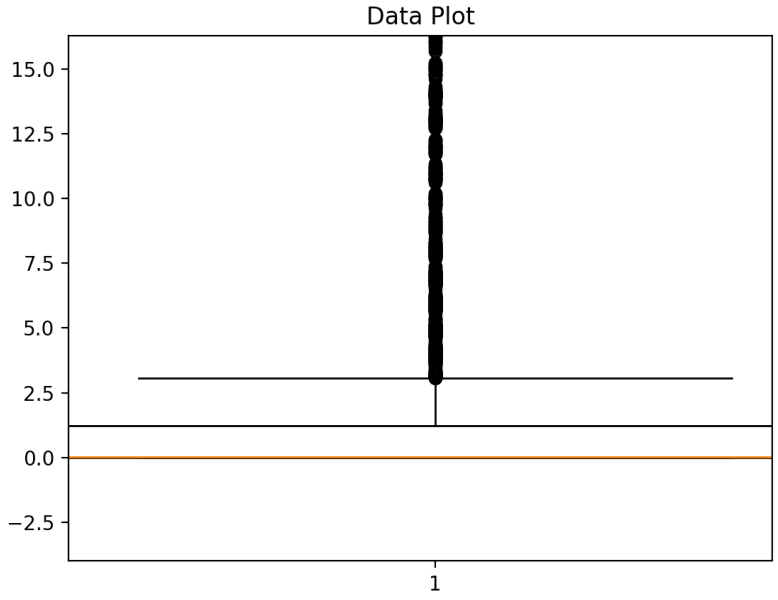

Figure 3.3: Boxplot of takt Time between completion of stories in days in the dataset used

3.4.4 Effort

In the effort approach, for each story of a project, since every project has its own particularities, we calculate, hour by hour of the time the story was in the state *"In Progress"*, how many stories was the developer assigned to this story, also assigned to and working on. This way we can assume a effort per hour on each past story of a project we are forecasting. We then use these efforts as historical data to run the Monte Carlo method and forecast the effort for future stories. The implementation is as follows.

```
1  def effort(idProject):
2     cursor.execute("SELECT a.date, c.date, b.id FROM changelog a INNER JOIN output b
           ON a.id = ANY(b.changelog) AND a.toString = 'In Progress' INNER JOIN
          changelog c ON c.id = ANY(b.changelog) AND c.toString = 'Done' AND b.project
          =" + str(idProject))
3     extraction = cursor.fetchall()
4     start= [i[0] for i in extraction]
5     finish = [n[1] for n in extraction]
6     tasksIds = [ti[2] for ti in tasksIds]
7     diff = [x2 - x1 for (x1, x2) in zip(start, finish)]
8     validation = []
9     startDates = []
10    idTasks = []
11    resolutionDates = []
12    minLimit = 300
13    maxLimit = 4320000
14    for index, x in enumerate(diff):
15       if(x.total_seconds() > minLimit and x.total_seconds() < maxLimit):
16          validation.append(x.total_seconds())
17          idTasks.append(tasksIds[index])
18          startDates.append(start[index])
19          resolutionDates.append(finish[index])
20     task = 0
21     historical = []
22     while task < len(idTasks):
23       cursor.execute("SELECT u.id FROM users u INNER JOIN output o ON u.id = o.
             assignee AND o.assignee IS NOT NULL AND o.id =" + str(idTasks[task]))
24       userID = cursor.fetchall()
25       if(len(userID) == 0):
26          task += 1
27          continue
28       cursor.execute("SELECT a.date, c.date FROM changelog a INNER JOIN output b ON
              a.id = ANY(b.changelog) AND a.toString = 'In Progress' INNER JOIN
             changelog c ON c.id = ANY(b.changelog) AND c.toString = 'Done' AND b.
             assignee=" + str(userID))
29       extractionT = cursor.fetchall()
30       startT= [i[0] for i in extractionT]
```

```
31      finishT = [n[1] for n in extractionT]
32      effortTask = 0
33      hour = startDates[task]
34      nhours = 0
35      while hour <= resolutionDates[task]:
36         tk = 0
37         counterTasks = 0
38         while tk < len(finishT):
39            if(hour <= finishT[tk] and hour >= startT[tk]):
40               counterTasks += 1
41            tk += 1
42         if(counterTasks > 0):
43            effortTask += 1/counterTasks
44         hour += datetime.timedelta(seconds=3600)
45         nhours += 1
46      hoursPerDay = 8 # 8 hours per day of work
47      totalEffort = (effortTask/nhours)*hoursPerDay #Total effort per story when
            working 8h per day
48      historical.append(totalEffort)
49      task += 1
50
51   return historical
```

In this implementation, after extracting the start date, as the moment the story was put in the state *"In Progress"* and the finish date as the moment that story was put in the state *"Done"*, we calculate the duration between these two moments for all the stories of one specific project. We also extract the ids of each of those stories. Since this first part is an integration of the initial part of the duration approach, we have the same limits for the difference between the start and finish of the story (Range between the dataset first quartile and the dataset third quartile). After having the stories who fit inside these limits, we iterate over all the stories. At each story, we extract the id of the user assigned to the story and then extract the start date, moment the story was put on the state *"In Progress"*, and finish date, moment the story was put on the state *"Done"*, of all the stories who have this user as its assigned developer. We then check hour by hour of the duration of the story we are analysing and count how many of the these stories with the same assigned developer were also on the state *"In Progress"*. After having the number of stories per hour, we divide this number per 1 hour, getting the effort per hour of the story we are analysing for all the hours of its duration. We then divide this total effort for the number of hours and multiply by 8, the assumed hours per day of work, this way, restricting the effort to only 8 hours per day. If one developer is only assigned to one story, the effort will be 8 hours per day, as expected. After having these efforts calculated to all the stories of the project selected, we return them as historical. For achieving the final delivery forecast, we need to simply use the assumption we made of having 8 hours of work per day and see how many days will it take to cover the effort forecast, divided by the number of developers available to work on the project. As explained above, due to the lack of information, non working days are not taken into account.

3.5 Forecasting Outputs and Charts

In this section we discuss the type of outputs and charts produced to show the results of the forecast. There are two types of outputs, the output of a forecast of a single story and the output of a the forecast of a set of stories, in this case, represented by charts.

3.5.1 Single Story Forecast

When forecasting for a single story, the output produced is a simple print of the values forecast. The model prints the id of the project, the point forecast (median) and the confidence interval forecast. Figure 3.4 shows the single story forecast output of an example using the duration approach. Figure 3.5 shows the output of the forecast of an example of one single story when using takt time to forecast. On Figure 3.6 is outputted the effort forecast for an example of one single story.

```
Project ID:711
Point Forecast: 7 days, 1:09:14
Interval Forecast: [23:19:20 -- 28 days, 1:33:40]
```

Figure 3.4: Example of an output of the duration approach for a single story

```
Project ID: 177
Point Forecast: 2:36:27
Interval Forecast: [0:06:19 -- 1 day, 21:32:09]
```

Figure 3.5: Example of an output of the takt time approach for a single story

```
Project ID:798
Point Forecast: 4:12:00
Interval Forecast: [1:20:00 -- 8:00:00]
```

Figure 3.6: Example of an output of the effort approach for a single story

As can be seen these outputs are very simple and due to being used on single story forecasts, don't have a lot of information for the user.

3.5.2 Set of Stories Forecast

When forecasting for a set of stories, the amount of information given to the user about the forecast can be more specific and allow the user to have a better perspective of the forecast for isolated stories in the middle of the set of stories. In order to produce an output with those characteristics, instead of simply printing the forecast of the full amount of stories, a chart is built where the forecast of each number of stories is represented, that is, for each number of stories till the final

forecast for *M* stories, a forecast is produced and represented in this chart output. Since when using duration for forecasting we get very high forecasts for just one single story, the utilization of this delivery forecasting approach to forecast for sets of stories was not pursued, due to the forecast of unrealistic high values. In the takt time approach, the model produces a chart where the historical data is represented and the forecast for the *M* number of stories to forecasts, starting after the last historical story, is also drawn. The forecast includes the median forecast and the confidence interval forecast. We consider pessimist forecast, the forecast made using the highest values inside the 95% confidence interval. Optimist is the forecast made using the lowest values inside the same 95% confidence interval. In this chart are also represented the completion date of the first story of the historical data and the completion date of the last story pessimist forecast. In between these extremes, the project finishing dates of the sprints that occurred and the ones that are planned to occur are also represented to give a better perspective of the number of stories that were and the ones the model forecast to be completed in each sprint.

In Figure 3.7 is represented an example of the output produced when using the takt time approach to forecast for a set of stories.

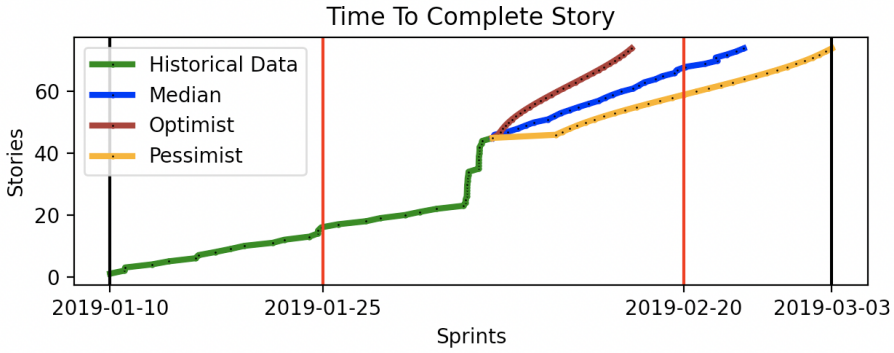

Figure 3.7: Example of an output of the takt time approach for a set of stories

When using the effort approach to forecast for a set of stories the model also produces a chart where the number *M* of stories to forecast are represented with the number of hours for completing that *M* stories. The chart includes the median forecast and the confidence interval forecast. Figure 3.8 represents an example of the chart produced when using effort to forecast to a set of stories. In this chart the historic data is not represented, contrary to what happens in the takt time chart, since in effort we don't consider the concept of continuity, that is, this chart represents the effort per number of stories, while in the takt time the chart, besides showing us the takt time between stories, also presents us the dates of sprints and delivery of the number of stories, using the concept of continuity to represent this dates.

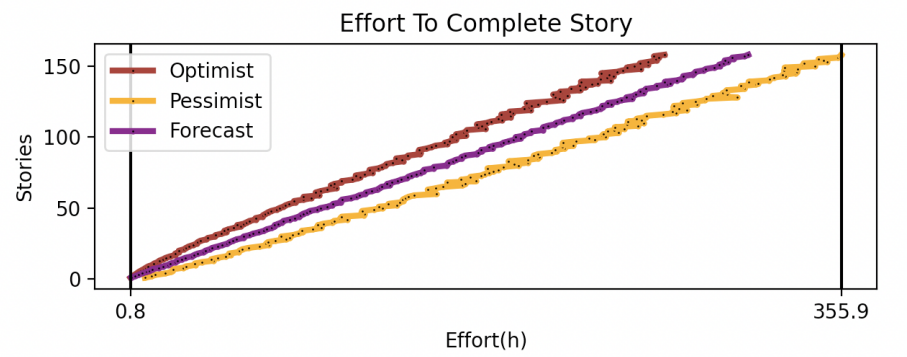

Figure 3.8: Example of an output of the effort approach for a set of stories

Chapter 4

Evaluation

In this chapter, we evaluate the delivery forecasting approaches mentioned in the implementation chapter. For each one of the approaches, we discuss positive and negative points, evaluate which approach provides better results in different situations and problems, and analyse comparisons with other existing models, this way proving the efficiency of the model and method.

We also validate the hypothesis proposed before, check and answer, based on the findings, each one of the research questions that help deconstruct the hypothesis.

4.1 Methodology

We evaluate the implementation described above by putting the model and each of the delivery forecasting approaches to test, against other tools and methods, revised in the previous chapters, using some of the most used evaluation metrics, already presented above. These metrics are the Magnitude of Relative Error (MRE), the Mean Magnitude of Relative Error (MMRE) , the Mean Root Square Relative Error (MRSRE) and the Percentage Error (PE).

For validating the hypothesis, we use the implementation and the Fraunhofer's Jira dataset. **Using this dataset, and after having the model implemented, the method and its different delivery forecasting approaches are compared with other methods to this way draw some conclusions about which provides better accuracy**, being this one of the main contributions of this dissertation. We then validate the research questions, who serve as guidelines for the main hypothesis. To answer RQ1, we make a comparison between the method and other, in order to show its efficacy, in particularly its efficiency against expert opinion methods. To answer RQ2, we discuss the type of input all three approaches revised follow and analyse the specificities of each one. To answer RQ3, we validate the results of the experiences made and base ourselves in this findings to take some conclusions on which approach works better in which situation. With this three questions answered, we take some conclusions about the main hypothesis and show its validity.

At the end, with the hypothesis validated, we prove that data-driven methods, in this case the Monte-Carlo method, are well fit for being applied in agile projects development forecast, bringing benefits in terms of reduced forecasting errors and user's effort, in comparison with other existing methods. **This is important because it draws conclusion who set the ground for future work**, another of the contributions of this dissertation.

4.2 Dataset

The dataset, whose structure was presented above, was provided by Fraunhofer, since Fraunhofer is also contributing to the PROMESSA project, which context this dissertation falls in.

This dataset contains 71 projects, each one with several stories, having the biggest project in the dataset around 2000 stories, what will be helpful to develop some experiments. It has 121 users who take actions on the projects, 12435 stories divided by the 71 projects, 136555 changes made to stories and 148 sprints. The dataset contains data about stories and projects from 2013 till 2020.

In order to check if the duration and takt time metrics follow a normal distribution, normality tests were made to these two metrics used.

The results of the normality test of the duration metric, by the Shapiro-Wilk algorithm [35], are presented in Table 4.1.

W	ρ	alpha level
0.897	4.77e-24	> 0.05

Table 4.1: Duration Shapiro-Wilk Normality Test

Since the ρ value is smaller than the alpha level of 0.05 chosen for the test, the hypothesis of the duration population being normally distributed is rejected, that is, the population doesn't follow a normal distribution.

The results of the normality test of the takt time metric, also by the Shapiro-Wilk algorithm [35], are presented in Table 4.2.

W	ρ	alpha level
0.041	0.0	> 0.05

Table 4.2: Takt Time Shapiro-Wilk Normality Test

Since the ρ value is also smaller than the alpha level of 0.05 chosen for the test, the hypothesis of the takt time population being normally distributed is also rejected.

In relation to the outliers identified, as shown by the boxplots represented in the previous chapter, the dataset has some data quality issues, as several unrealistic values are present. As shown in boxplot 3.2, stories with duration of 175 days don't represent realistic what happens in a project, and should be classified as data quality issues to resolve and, this way, removed from the

historical data. This is the main function of the limits, based on the first and third quartiles of the dataset, implemented in the previous chapter.

4.3 Evaluation Experiments

In this section we set up the experiences for the each one of the delivery forecasting approaches, implemented in the previous chapter, also presenting their results.

4.3.1 Duration Forecast For One Story

The duration approach to forecast delivery is the approach to forecast the time a story or a set of stories will take to be completed. As referred above, the duration is a forecast based on the time interval between the moment the story was put in the state *"In Progress"* till the moment the story is put in the state *"Done"*, since these are more accurate time windows than the one between the moment the story is created and the moment the story is completed. This happens because many time a story is created but only on later sprints is really worked on, leading to a time where the duration was already counting but in a reality the story was not being developed. Below is represented the setup of one experience made with the duration approach, that uses all the past stories of the project and that compares the forecasts with the real value of that story, that is, the real duration of the story to forecast.

```
1  def durationForecast(historical, realValue):
2      durationMC = monteCarlo(historical, 1)
3      median = durationMC[0]
4      lower = durationMC[2]
5      upper = durationMC[3]
6      real = realValue
7      upperEstimation = datetime.timedelta(seconds=upper)
8      upperEstimation = upperEstimation - datetime.timedelta(microseconds=
           upperEstimation.microseconds)
9      bottomEstimation = datetime.timedelta(seconds=lower)
10     bottomEstimation = bottomEstimation - datetime.timedelta(microseconds=
           bottomEstimation.microseconds)
11     rangeEstimation = "[" + str(bottomEstimation) + " -- " + str(upperEstimation) +
            "]"
12
13     actualValue = datetime.timedelta(seconds=real)
14     actualValue = actualValue - datetime.timedelta(microseconds=actualValue.
           microseconds)
```

In this experience we receive the historical data of the project where the story is inserted that was collected in the implementation shown on the previous chapter, and the real duration of the story for validating the accuracy of the model.

We call the Monte Carlo model, passing the historical data and the number of stories to forecast for, in this experiment set to 1 story. The model returns the median forecast and the upper and lower confidence limits. The range and median forecast are then printed.

Below are represented the results of some experiences of this experiment, using the duration to forecast delivery. The projects used one the experiences were selected randomly.

Proj	Hist	Forecast	95% Confidence Interval	Real Duration	RE
732	77	10 d, 20:28:22	[1d, 03:08:49 - 45 d, 04:06:59]	14 d, 08:47:48	24%
961	129	7 d, 00:59:36	[00:57:42 - 40 d, 02:39:17]	6 d, 23:59:54	1%
961	117	7 d, 00:44:05	[00:24:43 - 42 d, 01:16:55]	4 d, 02:26:35	71%
718	95	3 d, 21:51:30	[18:27:40 - 9 d, 00:14:48]	05:44:52	1536%
588	143	4 d, 19:31:49	[00:13:17 - 27 d, 18:34:29]	23:10:54	398%
688	35	10 d, 19:21:10	[01:17:54 - 29 d, 18:09:47]	17 d, 21:09:29	40%
652	81	6 d, 08:34:23	[00:09:04 - 39 d, 01:04:06]	6 d, 03:46:53	4%
887	17	05:51:38	[01:36:32 - 18 d, 03:37:13]	2 d, 04:58:56	88%
1090	29	20:54:46	[00:06:52 - 28 d, 00:05:51]	6 d, 03:17:25	86%
809	45	1 d, 03:49:10	[00:14:19 - 36 d, 07:05:47]	00:19:05	8646%

Table 4.3: Duration experiences

As can be seen, the duration approach produces confidence intervals too wide and most times not so accurate forecasts, as can be seen by the errors. This is due to the fact that even when the story is *"In Progress"*, the developing team is also working on other stories that may have higher priority what will lead to a extended duration when in reality the story was not being worked on. This will produce outputs as the ones seen in the Table 4.3 which are not very helpful for a team or developer trying to forecast how long will take the story to be completed.

Based on those outputs, we can take some considerations about the duration approach. Firstly this approach is very susceptible to errors, since we are using time intervals with a wide range of possible values. The duration of a story doesn't take into account the weekends, when nobody is working, and the amount of other stories open at the same time, that will make the story be *"In Progress"* longer that would be normal. Even though some degree of uncertainty is good for a forecast, since it will give the prediction some space for errors and delays, the uncertainty presented in this approach is too big, leading to a useless forecast.

In fact, this helps show why duration is not a good metric to use in forecasts, since it produces many times high values that will increase the noise inserted in the model, even if we remove outliers, as we did. We could say that this approach is also affected by the fact that is being used to forecast for a single story what will produce a bigger error than if used to forecast for a set of stories, but due to its wide confidence interval, when producing to a set of stories, the relative error would, probably, increase even more instead of compensate and get smaller.

4.3.2 Takt Time Forecast For A Set of Stories

Takt Time is a approach based on the time between completion dates of past stories. This approach is suitable to forecast the completion date for a set of stories since it analyses the time it passes between stories being completed.

The first experiment made with takt time was to forecast the delivery date for a set of stories and compare it with the values that occurred in real life. Below is represented the setup of this experiment.

```
def taktTimeForecast(historical, realValues, sprints, firstElement):
    y_axis_historical = []
    y_axis_forecast = []
    y = 1
    while y <= len(historical):
        y_axis_historical.append(y)
        y = y + 1
    while y <= len(realValues):
        y_axis_forecast.append(y)
        y = y + 1
    sum_historical = firstelement
    historical_sum = []
    t = 0
    while t < len(historical):
      sum_historical = sum_historical + datetime.timedelta(seconds = historical[t])
      historical_sum.append(sum_historical)
      t = t + 1
    medianForecast, lowerForecast, upperForecast = 0, 0, 0
    upper_sum, med_sum, val_sum, lower_sum = [], [], [], []
    sum_val = sum_historical
    f = 0
    while f < len(realValues):
      taktTimeMC = monteCarlo(historical, (f+1))
      medianForecast = sum_historical + datetime.timedelta(seconds = taktTimeMC[0])
      lowerForecast = sum_historical + datetime.timedelta(seconds = taktTimeMC[2])
      upperForecast = sum_historical + datetime.timedelta(seconds = taktTimeMC[3])
      sum_val = sum_val + datetime.timedelta(seconds = realValues[f])
      med_sum.append(medianForecast)
      upper_sum.append(upperForecast)
      lower_sum.append(lowerForecast)
      val_sum.append(sum_val)
      f = f + 1

  eL = 0
  error_median = []
  error_confidence = []
  outsideCounter = 0
```

```
38    while eL < len(realValues):
39       medianValue = med_sum[eL] - sum_historical
40       optimistValue = lower_sum[eL] - sum_historical
41       pessimistValue = upper_sum[eL] - sum_historical
42       pe_median = ((abs(realValues[eL] - medianValue))/realValues[eL]) * 100
43       error_median.append(pe_median)
44       if(realValues[eL] > pessimistValue):
45          error_confidence.append(((abs(realValues[eL].total_seconds() -
               pessimistValue.total_seconds()))/realValues[eL].total_seconds()) *
               100)
46          outsideCounter += 1
47       elif(realValues[eL] < optimistValue):
48          error_confidence.append(((abs(realValues[eL].total_seconds() -
               optimistValue.total_seconds()))/realValues[eL].total_seconds()) * 100)
49          outsideCounter += 1
50       else:
51          error_confidence.append(0)
52       eL = eL + 1
53
54    percentageOutside = (outsideCounter/len(error_confidence)) * 100
55    print("Stories outside Confidence Interval: " + str(outsideCounter) + "/" + str(
          len(error_condidence)) + "(" + str(round(percentageOutside, 2)) + "%)")
56
57
58
59    sprints = [spr for spr in sprints if spr <= max(max(upper_sum),max(val_sum))]
60    sprints = [spt for spt in sprints if spt >= firstElement]
61    sprints = list(dict.fromkeys(sprints))
```

This experiment receives the historical data, the real values of the tasks to forecast, the finishing dates of the sprints and the date of the first element of the project. The function first creates the y axis for the historical data and then for the stories to forecast. Then the function sets the x axis of the historical values. Since these values are takt times, we use a auxiliary variable *sum_historical* to starting from the date of the first element get the dates where each historical story was completed. The function then calls the Monte Carlo model and retrieves from it the median, upper and lower forecasts for the specific amount of tasks. This process is done for every number of tasks. Since the model returns takt times, the *sum_historical* is used to calculate the completion dates of the stories. The same thing is done to the set of real values, getting the real completion dates by using the adding the real takt times to a date auxiliary variable *sum_val*, that work as the *sum_historical*. After having all the forecasts and real values ready to be represented on a chart, we calculate the errors for each forecast, extracting first the date, in order to compare takt times and not dates. After this, we prepare the sprints by removing sprint dates that are beyond the first date and after the last real date or pessimist date. We then generate the charts with all this information.

In Figure 4.1 is presented and example of this experiment.

In this experiment, we take the takt time historical data and predict for the following stories. The number of stories and the finishing dates of sprints are also represented. The errors are represented on the second chart.

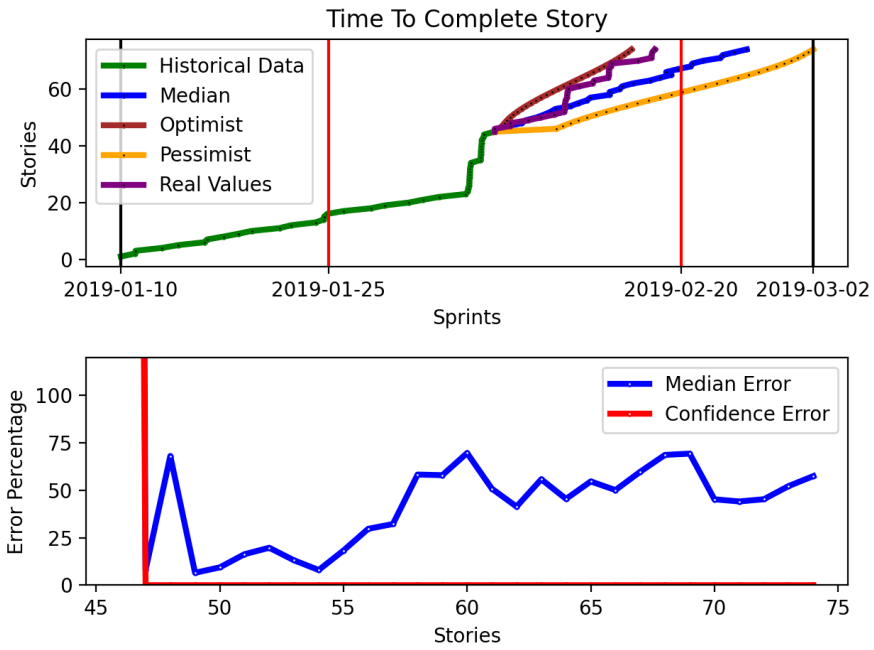

Figure 4.1: Takt Time Approach Output Example

The model presents to the user the chart forecast. The green line represents the previous stories takt times since the completion date of the first one of those elements. These stories will be used by the Monte Carlo method to forecast the following stories. The blue line represents the model forecast. The brown and yellow lines represent, respectively, the Optimist and Pessimist forecasts for each story. The sprints of the project are represented by the red lines, being the black lines the completion date of the first story and the pessimist forecast of the last story. The purple line represents the real takt times and completion dates of the stories to forecast. In the second chart outputted, we can analyse the percentage error of the forecast for each story predicted and the confidence error, that is, the percentage error of the real values in relation to the confidence interval.

We made 10 different experiences similar to the described, presented in Table 4.4, represented in days to the final delivery date.

The final MMRE for this 10 experiments was 32%, suggesting, this way, that this approach can be applied in the industry to forecast for the following sprints of a project, taking into account the past sprints of that project.

Proj	Hist	Forecast	95% Confidence Interval	Real Delivery	MRE
454	116	48.8	[39.4 - 56.4]	57.4	39%
961	49	27.5	[22.8 - 38.4]	22.4	29%
459	221	213.5	[193.7 - 230.1]	204.4	20%
578	86	35.4	[27.8 - 42.5]	33.1	27%
113	129	43.5	[33.1 - 51.1]	48.9	37%
798	101	34.5	[26.5 - 43.5]	37.9	26%
688	141	96.8	[82.4 - 110.4]	89.5	43%
934	8	7.2	[3.5 - 10.4]	6.5	21%
590	77	32.5	[25.1 - 39.8]	29.6	35%
725	114	88.1	[76.7 - 99.2]	92.4	38%

Table 4.4: Takt Time Experiences represented in days

4.3.3 Takt Time Historical/Forecast

The second experiment made with the takt time was in order to know, in this context, what were the best numbers of stories to use as historical data and as forecast. In this experiment we want to know the percentage error of the forecasts in relation to the number of stories to use as historical data and the number of stories to forecast. To aggregate the projects per historical and future number of stories is used the median of relative error or percentage error. Below is represented the setup of this experiment.

```
1  def histVSfor():
2     cursor.execute("SELECT id FROM projects")
3     idProjects = cursor.fetchall()
4     idProjects = [ip[0] for ip in idProjects]
5     pe_n = []
6     n_num = [5,10,15,20,25,30,35,40,45,50,55,60,65,70]
7     m_num = [5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]
8     nCount = 0
9     while nCount < len(n_num):
10       pe_mPoints = []
11       mCount = 0
12       while mCount < len(m_num):
13          median_pe = []
14          for project in idProjects:
15             cursor.execute("SELECT resolutionDate FROM output WHERE resolutionDate
                   IS NOT NULL AND project =" + str(project))
16             taskList = cursor.fetchall()
17             if(len(taskList) < 2):
18                continue
19             taskList = [tl[0] for tl in taskList]
20             taskList.sort()
21             projectTT = taktTime(project)
```

```
            firstelement = taskList[0]
            if(len(projectTT) < (n_num[nCount] + m_num[mCount])):
               continue
            historical = projectTT[:int(n_num[nCount])]  #First Half
            projectTT = projectTT[int(n_num[nCount]):]
            validation = projectTT[:int(m_num[mCount])]  #Second Half
            if len(historical) < 1:
               continue
            sum_historical = firstelement
            t = 0
            while t < len(historical):
               sum_historical = sum_historical + datetime.timedelta(seconds =
                   historical[t])
               t = t + 1
            medianForecast,lowerForecast, upperForecast = 0,0,0
            med_sum, val_sum, upper_sum,lower_sum = [], [], [], []
            f = 0
            sum_val = sum_historical
            while f < m_num[mCount]:
               taktTimeMC = monteCarlo(historical, (f+1))
               medianForecast = sum_historical + datetime.timedelta(seconds =
                   taktTimeMC[0])
               sum_val = sum_val + datetime.timedelta(seconds = validation[f])
               val_sum.append(sum_val)
               lowerForecast = sum_historical + datetime.timedelta(seconds =
                   taktTimeMC[2])
               upperForecast = sum_historical + datetime.timedelta(seconds =
                   taktTimeMC[3])
               med_sum.append(medianForecast)
               upper_sum.append(upperForecast)
               lower_sum.append(lowerForecast)
               f = f + 1
            eL = 0
            while eL < m_num[mCount]:
               realValue = val_sum[eL] - sum_historical
               medianValue = med_sum[eL] - sum_historical
               median_pe.append(((abs(realValue.total_seconds() - medianValue.
                   total_seconds()))/realValue.total_seconds()) * 100)
               eL = eL + 1
         pe_median = statistics.median(median_msre)
         pe_mPoints.append(rmsre_median)
         mCount += 1
      pe_n.append(pe_mPoints)
      nCount += 1
```

This experiment first extracts all the ids of the all the projects, then by each N number of historical stories and by each M number of stories to forecast, the function goes project by project and passes to the model N historical stories and the number of stories to forecast till it reaches M

stories. The function then calculates the error between these forecasts and the real values. The median of these errors is made, in order to aggregate all the project's errors. In the end a chart is produced showing the error in order of N and M.

In Figure 4.2 are presented the results of this second experiment with takt time. In order to allow better visibility and to be easier to understand each line relative to the values of historical data to use, two different charts were produced. For every project of the dataset, the percentage error of the forecasts in relation to the number of stories to use as historical data and the number of stories to forecast is shown.

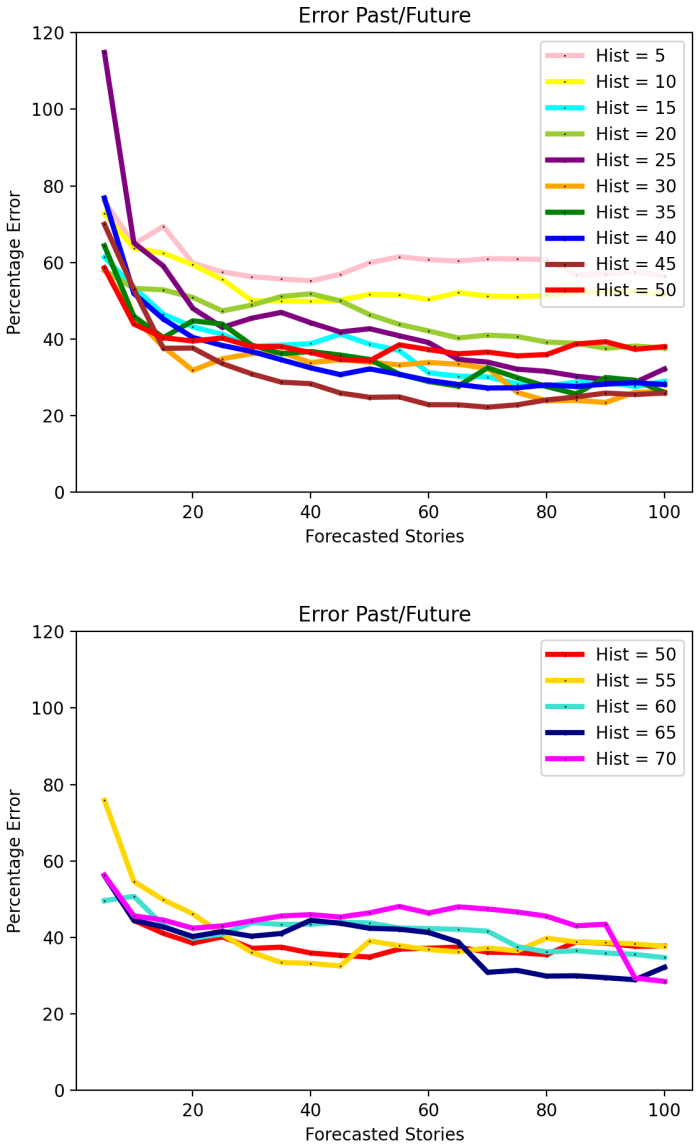

Figure 4.2: Ideal Historical/Forecast

Theses two charts represent, respectively, the number of historical stories from 5 to 50 and

from 50 to 70. As can be seen, using only 5 or 10 stories as historical data provides bigger errors, so we can say that in projects with less than 15 stories, the model will not be so accurate as in the case with more stories as historical. We can also note that when using 60, 65 or 70 stories as historical will also increase the error of the forecast, probably because of more noise added into the model. As we can notice, the ideal number of historical stories in this context, is when using 45 stories. The "Hist = 45" line provides better results than the others for most of the number of stories to forecast, getting to its lowest point in the 60 stories, that is, the lowest value of percentage error is achieved when forecasting 60 stories. With this in mind the ideal situation for the model would be when using 45 stories as historical stories to forecast 60 stories into the future. The model, as stated before, presents better results when using more than 10 stories as historical data. It could also be said that the model presents better results when forecasting for more than 20 stories, since after this value the errors start to compensate and balance each other. Is also expected that, when forecasting too much into the future, the error will start to increase again since the further we look into the future, more unpredictable that future is.

4.3.4 Takt Time Moving Window

The third and last experiment made with takt time was in order to take advantage of the fact that the dataset has a project with over 2000 stories, that when taking out the outliers goes to almost 900 stories. We analysed how does this delivery forecasting approach behaves with the evolution of the project. Below is presented the setup of a moving window that advances 60 stories at each iteration and uses the previous 45 stories to forecast for that window of 60 stories.

```
1  def movingWindow():
2     project_PE = []
3     project = 177
4     n = 45
5     m = 60
6     x_axis = []
7     cursor.execute("SELECT resolutionDate FROM output WHERE resolutionDate IS NOT
          NULL AND project =" + str(project))
8     taskList = cursor.fetchall()
9     if(len(taskList) < 2):
10      continue
11    taskList = [tl[0] for tl in taskList]
12    taskList.sort()
13    projectTT = taktTime(project)
14    projectLen = len(projectTT)
15    firstelementCounter = n
16    lastElementChecked = n + m
17    pe_window = []
18    projectX_axis = []
19    verical_lines = []
```

```
20    while(lastElementChecked <= projectLen):
21       firstelement = taskList[firstelementCounter]
22       if len(projectTT) < n:
23         break
24       historical = projectTT[:n]  #First Half
25       projectWithoutFH = projectTT[n:]
26       if len(projectWithoutFH) < m:
27         break
28       validation = projectWithoutFH[:m]  #Second Half
29       if len(historical) < 1:
30         break
31       sum_historical = firstelement
32       t = 0
33       while t < len(historical):
34         sum_historical = sum_historical + datetime.timedelta(seconds = historical[t
             ])
35         t = t + 1
36       medianForecast,lowerForecast, upperForecast = 0,0,0
37       med_sum, val_sum, upper_sum,lower_sum = [], [], [], []
38       f = 0
39       sum_val = sum_historical
40       while f < m:
41         taktTimeMC = monteCarlo(historical, (f+1))
42         medianForecast = sum_historical + datetime.timedelta(seconds = taktTimeMC
             [0])
43         sum_val = sum_val + datetime.timedelta(seconds = validation[f])
44         val_sum.append(sum_val)
45         lowerForecast = sum_historical + datetime.timedelta(seconds = taktTimeMC
             [2])
46         upperForecast = sum_historical + datetime.timedelta(seconds = taktTimeMC
             [3])
47         med_sum.append(medianForecast)
48         upper_sum.append(upperForecast)
49         lower_sum.append(lowerForecast)
50         f = f + 1
51       eL = 0
52       median_pe = []
53       while eL < m:
54         realValue = val_sum[eL] - sum_historical
55         medianValue = med_sum[eL] - sum_historical
56         median_pe.append(((abs(realValue.total_seconds() - medianValue.
             total_seconds()))/realValue.total_seconds()) * 100)
57         projectX_axis.append(firstelementCounter + 1)
58         firstelementCounter += 1
59         eL = eL + 1
60       pe_window += median_pe
61       verical_lines.append(lastElementChecked)
62       lastElementChecked += m
63       projectTT = projectTT[m:]
```

```
64          n += 5
65      project_PE.append(pe_window)
66      x_axis.append(projectX_axis)
```

In this experiment, we define get the stories of project with id 177 and starting after the first 45 stories, the historical stories of the first window, we forecast for the following 60 and compare the errors. The function then advances 60 stories, building a new window. This new window uses the real values of the last 45 stories of the previous window and forecasts the next 60. This procedure is repeated until the end of the project is reached. A chart with all the errors per window is made.

The results of the this experiment, the chart of a moving window that advances 60 stories at each iteration and uses the previous 45 stories to forecast for that window of 60 stories, are presented below. The black lines are limiting the 60 stories forecast. The red line represents the beginning of the forecast part of the first window, starting after the first 45 stories.

In the Figure 4.3 we can see that a forecast starts with a high error and then gets lower at almost every window, increasing rapidly to the start of the next window, proving this way what was stated before, that the percentage error gets lower when forecasting for more stories, even when forecasting in the middle of a project.

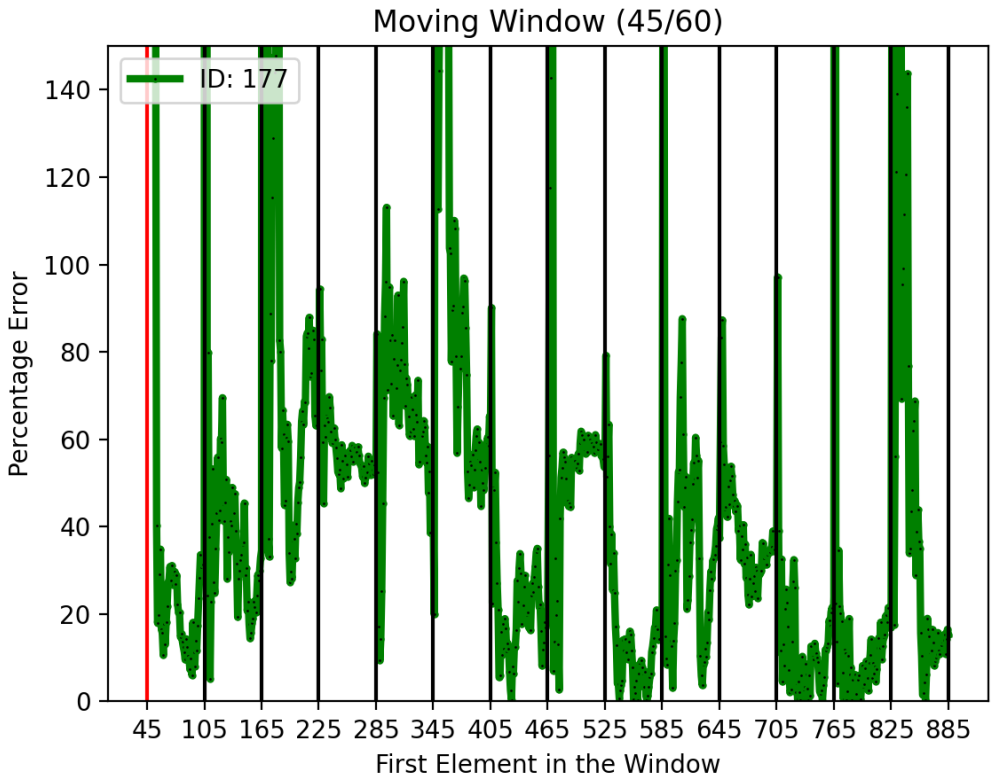

Figure 4.3: Moving Window for Project 177

4.3.5 Comparison with Sevawise

In the tables below we can check 4 experiences of the Takt Time approach when compared to the Sevawise's Monte Carlo model[1], mentioned above. The Sevawise model produces 5 outputs, instead of the 3 produced by our model. Sevawise implements the degrees Very Optimist, Optimist, Forecast, Pessimist and Very Pessimist while our model only implements the Optimist, Forecast and Pessimist degrees. This tool also sets to accomplish different objectives, using different approaches.

Since this tool created by Sevawise uses the approaches of velocity and story count, and measures time with the metrics of sprints and program increment, a time window containing sprints, we adapted this tool to work with our takt time, since their story count approach provides a final throughput, a measurement that is the inverse of the final takt time to complete the same number of stories. With this in mind, our historical stories are the equivalent to their sprints, being the takt time between stories equivalent to the amount of stories per sprint on their model. Their number of sprint per program increment will be set to with our number of stories to forecast, that is, the final result for that program increment, containing n stories, is going to be the inverse of our sum of takt time of n stories. With this mapping between tools, we can compare our method with theirs. Table 4.5 presents the 4 experiences made.

In these 4 experiences we can see that our model produces forecasts relatively close to the real value of time to develop a certain number of stories and also close to the forecasts produced by the Sevawise model. When evaluating our takt time approach in terms of RMSRE, for these 4 experiences, the values are, respectively, 55.27%, 64.98%, 49.55% and 16.16%. In terms of values outside the confidence interval in relation to our model's forecast, the values are, respectively, 4/19(21.05%) ,0/4(0.0%), 1/30(3.33%) and 0/124(0.0%), proving, this way, that even if the forecast is sometimes a bit far from the real value, the confidence interval forecast includes most of the real takt times for each story, helping a team or a developer understand better the context of the project and stories.

[1] More information about Sevawise tool can be found at https://sevawise.com/tools/monte-carlo

Labels	Model Developed	Sevawise Model	Real Value
Very Optimist	-	2 days, 00:53:14	7 days, 17:34:50
Optimist	2 days, 04:29:05	4 days, 17:05:30	-
Forecast	6 days, 07:41:25	6 days, 12:09:09	-
Pessimist	10 days, 03:50:10	8 days, 11:37:05	-
Very Pessimist	-	12 days, 06:24:10	-

(a) With 45 of historical, Forecast 19

Labels	Model Developed	Sevawise Model	Real Value
Very Optimist	-	04:16:16	2 days, 23:45:40
Optimist	02:07:24	1 day, 04:17:44	-
Forecast	1 day, 14:42:16	2 days, 03:40:11	-
Pessimist	3 days, 05:11:34	3 days, 05:54:12	-
Very Pessimist	-	5 days, 10:27:31	-

(b) With 45 of historical, Forecast 5

Labels	Model Developed	Sevawise Model	Real Value
Very Optimist	-	10 days, 11:27:53	11 days, 19:21:31
Optimist	9 days, 15:36:51	15 days, 14:39:55	-
Forecast	17 days, 14:51:46	18 days, 18:33:06	-
Pessimist	23 days, 04:03:32	22 days, 02:37:49	-
Very Pessimist	-	28 days, 04:31:49	-

(c) With 45 of historical, Forecast 30

Labels	Model Developed	Sevawise Model	Real Value
Very Optimist	-	41 days, 01:03:22	56 days, 17:53:28
Optimist	41 days, 15:05:07	49 days, 16:55:40	-
Forecast	54 days, 18:53:59	54 days, 21:18:26	-
Pessimist	66 days, 11:30:34	60 days, 06:02:27	-
Very Pessimist	-	70 days, 01:19:39	-

(d) With 45 of historical, Forecast 124

Table 4.5: Proposed Model vs Sevawise

4.3.6 Effort Forecast For One Story

The effort approach is an approach that uses the time a developer really spent working on a story as the base to make the forecast. For all the duration a story was in the state *"In Progress"*, the method will check how many stories the developer assign to the story was also working on, making this way an estimation of how much effort the developer put in to finish the story. Since this approach checks the amount of stories being developed at the same time, unlike the duration approach where a lot of noise was inserted in the Monte Carlo Method, with effort we can make more accurate forecasts and reduce the huge uncertainty inherent to the duration approach. Below is represented the setup of the first experiment using the effort approach.

```
def effortForecast(historical, realValue, estimation):

    effortMC = monteCarlo(historical, 1)
    median = effortMC[0]
    lower = effortMC[2]
    upper = effortMC[3]
    real = realValue
    upperEstimation = datetime.timedelta(seconds=upper)
    upperEstimation = upperEstimation - datetime.timedelta(microseconds=
        upperEstimation.microseconds)
    bottomEstimation = datetime.timedelta(seconds=lower)
    bottomEstimation = bottomEstimation - datetime.timedelta(microseconds=
        bottomEstimation.microseconds)
    rangeEstimation = "[" + str(bottomEstimation) + " -- " + str(upperEstimation) +
         "]"

    originalEstimation = datetime.timedelta(seconds=estimation)
    originalEstimation = originalEstimation - datetime.timedelta(microseconds=
        originalEstimation.microseconds)

    actualValue = datetime.timedelta(seconds=real)
    actualValue = actualValue - datetime.timedelta(microseconds=actualValue.
        microseconds)
```

This experiment receives the historical data of the project, the real value of the story to forecast and the estimation made by Fraunhofer's developers for that story. We call the Monte Carlo model, passing the historical data and the number of stories to forecast for, in this experiment set to 1 story. The model returns the median forecast and the upper and lower confidence limits. The range and median forecast are then printed, as well as the real value and the estimation.

The results of this first experiment made with effort are represented in the Table 4.6, showing a set of experiences made with the effort approach, comparing the forecasts outputted to the real value of effort the story to forecast had and also comparing to the estimations made by Fraunhofer's developers, making this way a comparison of this method against expert opinion methods. The relative error or percentage error and the number of historical stories used in each forecast is also presented.

As can be seen in the table the model makes, most of the times, very accurate forecasts in relation to the real value, and when this doesn't happen, the confidence interval provides a small degree of uncertainty that in all this cases include the real value, giving this way a good perspective of the context of the story. We can also see that in most cases, with a great amount of historical stories, the model forecasts with greater accuracy. In comparison to the estimations made by developers, few are the cases where the expert opinion method makes better estimations than the model forecasts. The effort approach is, this way, a good alternative to the takt time when we want to forecast for one story. In terms of values inside the confidence interval, all the real values are within the range forecast.

Proj.	Hist	Forecast	Confidence Interval	User Estimates	Real Value	RE
798	86	04:00:00	[01:20:00 - 08:00:00]	20:00:00	04:00:00	0%
286	67	01:00:00	[00:54:43 - 01:46:34]	08:00:00	01:46:34	43%
798	34	04:00:00	[01:08:34 - 08:00:00]	01:00:00	08:00:00	50%
688	18	01:07:07	[00:43:38 - 07:59:34]	03:30:00	04:00:00	72%
840	78	08:00:00	[06:00:00 - 09:00:00]	01:00:00	07:49:05	4.5%
688	59	01:08:34	[00:43:44 - 06:46:36]	04:00:00	00:46:54	46%
818	63	04:00:00	[01:55:22 - 08:00:00]	01:00:00	01:57:13	50%
688	96	01:09:37	[00:43:44 - 07:06:10]	04:00:00	00:46:55	32%
688	123	01:05:15	[00:43:40 - 07:59:34]	12:00:00	01:13:56	10%
688	32	01:17:13	[00:43:49 - 08:00:00]	12:00:00	04:59:42	74%

Table 4.6: Effort Experience

4.3.7 Effort Forecast For A Set of Stories

The second experiment to do with the effort approach is to test its efficacy when forecasting for a set of stories. Below is represented the setup of that experiment.

```
1  def effortSet(historical, realValues, estimations):
2      r = 0
3      sum_real_effort = 0
4      sum_estimation = 0
5      real_val = []
6      med_val = []
7      lower_val = []
8      upper_val = []
9      error_val = []
10     confidence_val = []
11     estimation_val = []
12     y_axis = []
13
14     while r < len(realValues):
15         forecastMC = monteCarlo(historical, (r + 1) )
16         forecastMed = forecastMC[0]
17         lower = forecastMC[2]
18         upper = forecastMC[3]
19         sum_real_effort += realValues[r]
20         sum_estimation += estimations[r]
21
22         real_val.append(sum_real_effort)
23         estimation_val.append((sum_estimation/3600)) # Convert Estimation from hours
               to seconds
24         med_val.append(forecastMed)
25         upper_val.append(upper)
26         lower_val.append(lower)
```

```
27      y_axis.append((r+1))
28
29      if(sum_real_effort > upper):
30          confidence_val.append(((abs(sum_real_effort - upper))/sum_real_effort) *
                100)
31      elif(sum_real_effort < lower):
32          confidence_val.append(((abs(sum_real_effort - lower))/sum_real_effort) *
                100)
33      else:
34          confidence_val.append(0)
35      error_val.append(((abs(sum_real_effort - forecastMed))/sum_real_effort) *
            100)
36      r += 1
```

The experiment receives the historical data of the project, the real values of the stories to forecast and the estimations of those stories. First, the function calls the Monte Carlo method in order to get the median, upper and lower forecasts and then it compares the forecasts with the real values, getting the forecast error and confidence error of the model. A chart with these results is made in order to facilitate the visualization.

Figure 4.4 represents an example output of the chart produced.

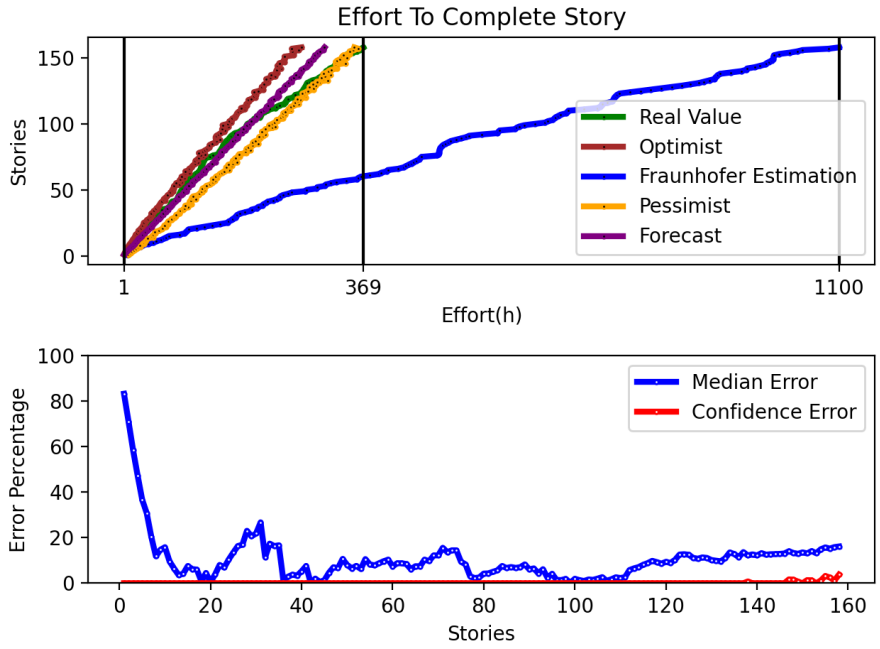

Figure 4.4: Example of the effort of a set of stories, comparing with estimation

We made 5 experiences that are represented in Table 4.7. We use half of the project as historical and the other half to validate the forecasts made.

Proj	Hist	For.	95% Conf. Int.	Real Effort	User Est.	For. MRE	Est. MRE
688	101	306h	[287h - 369h]	378h	1100h	20%	191%
798	16	75h	[55h - 94h]	68h	129h	11%	89%
840	12	95h	[83h - 103h]	59h	87h	31%	48%
1090	17	81h	[70h - 93h]	87h	237h	12%	172%
997	14	69h	[54h - 85h]	65h	177h	27%	172%

Table 4.7: Effort Experiences

The final MMRE for our model on this 5 experiences is 20%, while the final MMRE for the user's estimations on this 5 experiences is 134%.

As can be seen in the results of this experiment, the effort approach provides good results when forecasting for a set of stories.

Is also worth compare the estimations provided by the developers in relation to a set of stories with our effort approach and with the real value. As we can see, the estimations made by experts provide bad accuracy when forecasting for a set of stories.

In relation to our model, when we look too much into the future, in this case after around 100 stories, the forecast starts to get less accurate, mainly because of the high degree of uncertainty inherent to the further way we look into the future, as highlighted in the experience with project 688.

A problem with the effort approach is the fact that it's based on two assumptions. The first is saying a developer will work on all its assigned and opened stories during one hour. This will not be true most of the times, but since at the end the developer will work on all the stories, these assumptions will balance each other, normalizing again the error. The second assumption is that a developer works 8 hours per day.

The effort approach can also be used to make some team management in terms of people working on the project and, this way, make some forecasts about the project costs. In this context, since the dataset provided by Fraunhofer only has information about the developer assigned to the story, and not about all the members that, possibly, worked on the story, this team management and cost forecast could not be included and implemented in this approach.

4.4 Validation Threats

In this section, we identify possible reasons why the results presented could have flaws or might not be generalizable, and explain how we tried to prevent that.

One of the reasons that could lead to wrong results is the presence of unrealistic duration and takt times in the dataset which would increase the noise present in the model, leading to inaccurate

results. This noise is probably due to cases where developers close a lot of stories, that they had forget to close before, at the same time, cases where developers set a story as *"In Progress"* and moments after, they realize that the story was already concluded and close it in a small amount of time, and cases where developers forget to close the story and thus leading to unrealistic big amounts of time. We tried to solve this problem using the concept of outliers, values outside the range between the first and third quartile.

Another issue that could threat the legitimacy of the results is the assumptions made in the effort approach. Since we assume that the developer works the same amount of time per hour on all its assigned stories, what doesn't represent the reality, because what really happens is that a developer probably works some hours on one story and then shifts to another. Assuming that only the assigned developer works on a specific story could also be wrong, since sometimes other developers of the team responsible for the project where the story is inserted in can also work on the story, increasing the effort that will not be taking into account. We also assume that a developer only works 8 hours per day, another assumption that don't necessarily happens all the times, since sometimes developers do extra hours, increasing the hours per day. Since the real values of effort are also calculated using these assumptions, since no record of effort is provided by the dataset, this could explain the overestimates made by the estimation team of Fraunhofer. All these assumptions are made as a alternative to work with the information in the dataset, due to the lack of information about real effort and lack of information about which developers worked in the story, and because of this could represent threats to the legitimacy of the results presented.

4.5 Discussion

In this section, we are going to discuss the main conclusions achieved in the experiences and based on them validate the hypothesis proposed.

In order to validate the main hypothesis stated before, we should first answer the research questions that deconstruct this main hypothesis and discuss each one.

4.5.1 RQ1: Is the chosen method more efficient than the others?

Through the revision of the state of the art, we reach to the conclusion that the Monte Carlo method brings benefits in relation to others. Its applicability in agile context, its output, being easy to understand and implement, its granularity and the good accuracy it provides are some of those benefits.

In terms of efficiency when compared to other methods, in particularly when compared with expert opinion methods, the results achieved with the effort approach and experiments suggest that, in this context, the Monte Carlo method forecasts may provide better accuracy than the estimations made by experts, and has the major benefit of providing good forecast without the team or experts having to lose a lot of time doing the estimations, precious time that could be used in another tasks, a problem strongly highlighted by the #NoEstimates movement.

4.5.2 RQ2: Which type of data will this method work with as input?

The Monte Carlo method takes, in this context, time measurements. All the delivery forecasting approaches followed work with some type of time metric as input. For the duration approach, the model takes as input the time interval between the moment the story was put in the state *"In Progress"* till the moment the story is put in the state *"Done"*. For the Takt Time approach, the model takes as input the time intervals between conclusion of stories, forecasting this way the amount of time that will pass between the completion date of a set of stories. For the effort approach, the model takes as input the time that a developer spent working on a story, working in a similar way to duration with the exception that only the time that the story was really being worked on counts for the model.

4.5.3 RQ3: What approach would be more useful for better forecasts?

In the previous chapter, 3 delivery forecasting approaches were explained and implemented. The first one, duration, was, for us, the worst approach. Since we are looking at past stories and calculating the entire duration the story was in development, we are counting a number of hours where the story development was completely stopped. If we take into account that in a company a story will only be worked on during the week and, probably, 8 hours per day, we are counting a lot of hours where the story was not worked on. In addiction to that if we consider that other stories and projects are developed at the same time, even in the hours we could consider as active hours, a developer will not be working on this one story all the time. The biggest problem with all this added noise is that forecast will get bigger values and with this errors and accumulative errors will also increase. The degree of uncertainty implicit in the confidence interval will also be huge, since the confidence interval has a tremendous range. With a range of values so big and even bigger confidence interval, we can say that the forecast will have so much noise that it will be useless for a team or developer.

The second approach is takt time, an approach that instead of using times intervals of particular stories, uses the time intervals between the conclusion of those stories. This will allow for very accurate forecast when forecasting for bigger sets of stories since the errors will accumulate and balance each other, diminishing the error until a certain point in the future where the error will start to increase again due to the bigger uncertainty of a too far future. This approach is, this way, a good approach to use when wanting to forecast for an entire project or for a sprint or release, where we expect a certain number of stories that allow for good accuracy.

The third approach is the effort, an approach that starts from the duration approach and resolves some of the problems inherent to the former. This approach will hour by hour of the duration of a story, check how many stories were also opened and assigned to the developer assign to the story and this way calculate the time per hour a developer spent working on the story. Doing this for the entire duration of the story will give us the effort the developer assigned put in the story. This approach eliminates the problems of the duration, providing forecast with much less noise. The problem with this approach is the assumption it is based on. The big assumption here is saying that

in a specific hour, the developer worked on all its assigned stories, when in reality this is not true, but when looking at the final result, these assumptions will balance each other, since a developer will, in the end, work on all the stories even if not all in the same hour. Another assumption made is that a developer works 8 hours per day, what could not be true. This approach provides good results both when forecasting one single story as well as when forecasting a set of stories.

In conclusion, and based on the results achieved, the most useful delivery forecasting approaches are the takt time and effort. The takt time is the most useful and accurate when forecasting for a bigger set of stories and when forecasting delivery dates. The effort approach is the most useful and accurate, in relation to the other approaches, when forecasting for one story or a few but also provides accurate results when forecasting for bigger sets of stories, when forecasting effort.

4.5.4 Hypothesis

After the extensive analysis of all the results obtained and calculated, allow us to discuss the the validity of the hypothesis, taking into account some observations taken during the experiments.

As verified in the state of the art chapter, there are a lot of data-driven methods who work with different metrics. We choose the Monte Carlo simulation method, to help prove that this type of methods can achieve better accuracy and efficiency than expert-opinion based methods.

In terms of cost and time efficiency, data-driven methods achieve forecasts in a much smaller amount of time and with lesser costs, since this type of forecast only requires computer processing power, something that nowadays is of easy access, and no team or experts are required to spent time on this task, time that in a company is also synonym of money, since the developers or experts are getting paid for that time.

In terms of accuracy, based on some of the findings highlighted in the state of the art chapter and in the findings achieved after implementing the model, that allowed for more practical tests against expert-opinion methods, we can conclude that data-driven methods, in this case the Monte Carlo method, can achieve forecasts with better accuracy than expert-opinion estimations. The approach that better shows this statement is the effort approach, since it allowed for a direct comparison with the expert-opinion estimations provided by the experts and developers. On that context, is safe to affirm that this data-driven model provided much more accurate forecasts than the developers estimations. This is even more clear when comparing the forecasts for various stories of one project against the estimations for the same amount of stories provided by developers, as can be seen in Figure 4.4.

Based on this analyses, we verify the veracity of the hypothesis, stating that data-driven methods for software project forecasting can, as suggested by the results achieved, with a good level of accuracy, be applied in agile projects, bringing benefits in terms of reduced forecasting errors and user's effort, in particularly when comparing with expert-opinion methods.

Chapter 5

Conclusion

The document presented, analyses and reports important notions about this field, presents a detailed report of some of the most used metrics, both for measurement of input/output and for validation. It analyses some of the most used methods and reports their major benefits, limitations and make a comparison between them, highlighting the most promising methods.

After selecting the Monte Carlo method, due to its simple applicability in agile context, it's understandability and it's granularity, a forecasting model was implemented. Having 3 different delivery forecasting approaches implemented, in order to evaluate, we recurred to the Fraunhofer Jira dataset to build some experiments. With this evaluation of each approach, we discuss some of the negative and positive points of each one of those approaches and in which context each one achieves better results. With the results obtained from the experiments made, we were able to discuss each alternative and finally demonstrate the hypothesis stated in the beginning of the dissertation.

The tool implemented is ready to be deployed and applied in real life context forecasting. The tool is working as a shell script, having a basic interface of simple prints and charts. **This tool can be use by developers to forecast for one single story or for a set of stories**, being this one of the contributions of the dissertation. The most recommended approaches to use are the takt time and effort approaches, each one having its particularities, as discussed above. The tool is implemented to work with the jira database structure, so in order to use the tool with other structures, some changes in the tool must be done, something that is facilitated by its modular structure.

Some difficulties were encountered when trying to build the forecasting model to produce accurate forecasts, since the dataset had a lot of unrealistic duration and takt time values, something that is very representative of what happens in the industry. We had to find techniques, like the usage of quartiles to assume realistic limits, that would allow us to remove these unrealistic values in order to have a less noisy dataset, something critical to achieving accurate forecasts.

In the future, an implementation of a machine learning method could also be relevant, since this type of methods also have a lot of positive points that can help providing more accurate results,

like the possibility of discover existing patterns between projects, the avoidance of overfitting to data, that is, the ease to adapt to new situations, something that doesn't happen with the Monte Carlo method, and the generalization capability. Some of the findings found using the Monte Carlo method, are also relevant for the Machine Learning implementation, such as the fact that duration is not a very accurate delivery forecasting data feature due to its accumulation of noise present in the dataset. Takt Time and Effort are, based on the findings, two good delivery forecasting data features to use on this Machine Learning implementation, due to the fact that they provide less noisy data values to the model, allowing it to achieve accurate forecasts. That implementation could be used to compare with the results found in this dissertation, in order to show the benefits of machine learning methods, stated above, in relation to the Monte Carlo method. This implementation would also help proving the efficacy and accuracy of data-driven methods.

The integration of this model and of the machine learning model with a project development management tool, like Jira, could also be very helpful since it will allow for a more in depth empirical study with users and compare the forecast against their estimations. This integration would also allow for the development of a better interface design. An option for forecasting should be included in the project dashboard, so that users could use this tool in their projects to forecast future sprints, team efforts and delivery dates. This plugin would allow for a data collection that could be use to access the model accuracy and efficiency, also proving the benefits of data-driven methods forecast. Since the tool was design to work with the jira database structure, the integration on this project development management tool would be easier to do.

References

[1] Mohamed Abouelela and Luigi Benedicenti. Bayesian network based xp process modelling. *International Journal of Software Engineering & Applications*, July 2010.

[2] Deepak Ahlawat and Rshma Chawla. Software development effort estimation using fuzzy logic framework - an implementation. *IRD India*, June 2015.

[3] Iman Attarzadeh and Siew Hock Ow. Software development effort estimation based on a new fuzzy logic model. *International Journal of Computer Theory and Engineering*, October 2009.

[4] Thomas Betts. Web-based monte carlo simulation for agile estimation. Available at https://mobilemonitoringsolutions.com/web-based-monte-carlo-simulation-for-agile-estimation/, 2019.

[5] Roheet Bhatnagar, Vandana Bhattacharjee, and Mrinal Kanti Ghose. Software development effort estimation – neural network vs. regression modeling approach. *International Journal of Engineering Science and Technology*, 2010.

[6] Pawel Brodzinski. Estimation and forecasting. Available at https://www.agilealliance.org/estimation-and-forecasting//, 2015.

[7] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1995.

[8] Stipe Celar, Srdjana Dragicevic, and Mili Turicc. Bayesian network model for task effort estimation in agile software development. *The Journal of Systems and Software*, January 2017.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 1995.

[10] Wei Lin Du, Danny Ho, and Luiz Fernando Capretz. A neuro-fuzzy model with seer-sem for software effort estimation. In *25th International Forum on COCOMO and Systems/Software Cost Modeling*, 2010.

[11] Richard E. Fairley. Recent advances in software estimation techniques. In ACM, editor, *ICSE '92: Proceedings of the 14th international conference on Software engineering*, page 382–391. ACM, 1992.

[12] Adrian Fittolani. Agile project forecasting – the monte carlo method. Scrumage. Available at http://scrumage.com/blog/2015/09/agile-project-forecasting-the-monte-carlo-method/, 2015.

[13] Andrew Gray and Stephen G. MacDonell. Applications of fuzzy logic to software metric models for development effort estimation. In Computer Society Press, editor, *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society*, pages 394–399. IEEE, 1997.

[14] James Grenning. Planning poker, April 2002.

[15] Omar Hidmi and Betul Erdogdu Sakar. Software development effort estimation using ensemble machine learning. *Journal of Computing, Communications & Instrumentation Engineering*, March 2017.

[16] Sultan Mohiuddin Hydarali. *Comparison of Artificial Neural Network and regression models for estimating Software development effort*. PhD thesis, University of Limerick, 2019.

[17] Ali Idri, Fatima azzahra Amazal, and Alain Abran. Analogy-based software development effort estimation: A systematic mapping and review. *Elsevier*, July 2014.

[18] Ali Idri, Taghi M. Khoshgoftaar, and Alain Abran. Can neural networks be easily interpreted in software cost estimation? In IEEE, editor, *2002 IEEE World Congress on Computational Intelligence*. IEEE, 2002.

[19] Ali Idri, Taghi M. Khoshgoftaar, and Alain Abran. Estimating software project effort by analogy based on linguistic values. In IEEE, editor, *Proceedings Eighth IEEE Symposium on Software Metrics*. IEEE, 2002.

[20] Galorath Incorporated. *SEER-SEM Product Brief*. Galorath Incorporated, 2011.

[21] Ron Jeffries. The noestimates movement. Available at https://ronjeffries.com/xprog/articles/the-noestimates-movement/, 2013.

[22] Magne Jørgensen. Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting 23*, 2007.

[23] Magne Jørgensen, Barry Boehm, and Stan Rifken. Software development effort estimation: Formal models or expert judgment? *IEEE Software*, May 2009.

[24] Magne Jørgensen, Tore Dybå, and Helen Sharp. What we do and don't know about software development effort estimation. *Voice of Evidence*, April 2014.

[25] Liz Keogh. The estimates in #noestimates. Available at https://lizkeogh.com/2015/05/01/the-estimates-in-noestimates/, 2015.

[26] Ekrem Kocaguneli, Ayse Tosun, and Ayse Bener. Ai-based models for software effort estimation. In IEEE, editor, *36th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2010.

[27] Troy Magennis. Managing software development risk using modeling and monte carlo simulation. In *Lean Software and Systems Conference 2012*, 2011.

[28] Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, 2006.

[29] Darko Milicic. *Applying COCOMO II - A case study*. PhD thesis, Blekinge Institute of Technology, 2004.

[30] Dan North. Monte python simulation: Misunderstanding monte carlo. Available at https://dannorth.net/2018/09/04/monte-python-simulation/, 2018.

[31] James Rodger Parag Pendharkar and Girish Subramanian. A probabilistic model for predicting software development effort. *IEEE Transactions on Software Engineering 31*, July 2005.

[32] Mirko Perkusich, Hyggo Oliveira De Almeida, and Angelo Perkusich. A model to detect problems on scrum-based software development projects. In ACM, editor, *SAC '13: Proceedings of the 28th Annual ACM Symposium on Applied Computing*, page 1037–1042. ACM, 2013.

[33] Shashank Mouli Satapathy, Aditi Panda, and Santanu Kumar Rath. Story point approach based agile software effort estimation using various svr kernel methods. In SEKE, editor, *The 26th International Conference on Software Engineering and Knowledge Engineering*. SEKE, 2014.

[34] Christopher Schofield. *An Empirical Investigation into Software Effort Estimation by Analogy*. PhD thesis, Bournemouth University, 1998.

[35] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52:591–611, 1965.

[36] Vishal Sharma and Harsh Kumar Verma. Optimized fuzzy logic based framework for effort estimation in software development. *International Journal of Computer Science Issues*, March 2010.

[37] Alberto Sillitti, Giancarlo Succi, Witold Pedrycz, Raimund Moser, and Pekka Abrahamsson. Effort prediction in iterative software development processes – incremental versus global prediction models. In IEEE, editor, *First International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2007.

[38] Amit Sinhal and Bhupendra Verma. Software development effort estimation: A review. *International Journal of Advanced Research in Computer Science and Software Engineering*, June 2013.

[39] Andrew Stellman and Jennifer Greene. Applied software project management: Estimation. Available at https://www.stellman-greene.com/LectureNotes/03%20estimation.pdf.

[40] Derya Toka and Oktay Turetken. Accuracy of contemporary parametric software estimation models: A comparative analysis. In IEEE, editor, *39th Euromicro Conference Series on Software Engineering and Advanced Applications*. IEEE, 2013.

[41] Muhammad Usman, Emilia Mendes, Francila Weidt, and Ricardo Britto. Effort estimation in agile software development: A systematic literature review. In ACM, editor, *PROMISE '14: Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, page 82–91. ACM, 2014.

[42] Juanjuan Zang. Agile estimation with monte carlo simulation. In Springer, editor, *Agile Processes in Software Engineering and Extreme Programming*, pages 172–179. Springer, 2008.