POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

Master's thesis

ANALYSIS OF CONCEPT DRIFT IN EFFORT ESTIMATION FOR IT PROJECTS

Julia Mularczyk, 148062

Supervisor
dr hab. inż. Mirosław Ochodek

POZNAŃ 2025

Abstract

Background: Effort estimation is a fundamental activity in software engineering, as it provides the basis for scheduling, budgeting, and resource allocation. Yet despite decades of research, estimation accuracy remains limited, particularly in Agile contexts where requirements evolve rapidly and uncertainty is high. From a data-driven perspective, such changes can be interpreted as concept drift – the shift of statistical properties in project data over time – which may cause predictive models to lose validity.

Goal: This thesis investigates the role of concept drift in effort estimation for IT projects, with a particular focus on the task level, where estimates are given for user stories and small work items. At this level, estimation is especially challenging due to variability in scope and developer productivity, and little research has explored the effect of concept drift. Two research questions guide the study: whether concept drift can be detected in effort estimation datasets, and whether existing adaptive strategies can mitigate its impact.

Method: The research relies on the TAWOS dataset of Agile projects. A Random Forest model is used as the baseline, with preprocessing steps including text embeddings of issue descriptions and categorical encodings. The experimental procedure is divided into three parts: temporal folds to establish baselines across project histories, sliding window analysis to illustrate the effect of outdated training data, and drift detection using ADWIN. In the adaptive setup, retraining is performed after drift detection with a weighted combination of old and new models. Evaluation is based on common metrics (MAE, MSE, R^2, MMRE) and local error comparisons before and after drift points.

Results: The experiments show that concept drift is a recurring phenomenon in effort estimation. Temporal folds revealed that models trained on older data performed worse than those using recent history, with MAE reductions of over 30% in several projects. Sliding windows highlighted episodic drift, with abrupt accuracy drops and recoveries, such as $\mathbf{R^2}$ shifting from negative to above 0.4 between consecutive windows. ADWIN detected multiple drift points in larger datasets; in Project 28, retraining reduced MAE from 6.45 to 0.56 and from an outlier of 874 to just over 1. In several cases retraining reduced error substantially, such as in Project 11 where MAE fell from 1.92 to 0.42, though in some projects performance worsened, showing that adaptation is not universally reliable.

Conclusions: The findings contribute an empirical analysis of concept drift in software effort estimation. While not providing final solutions, the experiments offer insights into the dynamics of model degradation and adaptation, laying the groundwork for more robust estimation techniques in evolving project environments.

Keywords: effort estimation, concept drift, machine learning, task-level estimation, ADWIN, software engineering, Agile projects

Contents

Chapter 1

Introduction

Project planning is one of the most critical activities in software development projects. It provides the foundation for scheduling, budgeting, and allocating resources, and thus has a decisive influence on project success [1]. A key component of planning is effort estimation, the process of predicting the amount of work required to complete a project or a given task [2]. Reliable estimates reduce the risk of budget overruns and missed deadlines, while inaccurate ones often lead to serious organizational and financial consequences [2].

Over the years, various methods have been applied to support effort estimation. Traditional approaches include expert judgement, analogy-based reasoning, and algorithmic models such as COCOMO [1]. More recently, machine learning and other data-driven techniques have been used to improve prediction accuracy [3, 4, 5]. Estimation is performed not only at the level of entire projects but also within smaller planning cycles, such as iterations or sprints in Agile methodologies, and even at the level of individual tasks [6, 7]. At these finer levels, the problem becomes even more challenging due to high variability and uncertainty [5].

Despite these methodological advances, recent studies confirm that estimation challenges remain. The replication studies by Tawosi, Moussa, and Sarro [6, 7] demonstrate that even in Agile environments – where effort is commonly estimated using story points and iterative planning – accuracy is still limited and results vary significantly across projects. This underlines that the problem of effort estimation is far from solved and highlights the need to investigate additional factors, such as concept drift, that may explain why predictive performance deteriorates over time.

Another source of difficulty is the dynamic nature of IT projects. Requirements change, technologies evolve, and teams adapt to new practices [5]. From the perspective of data analysis, such changes can be interpreted as concept drift – a shift in the statistical properties of input or output variables over time [8, 9, 10]. When concept drift occurs, predictive models trained on past data gradually lose their accuracy unless they are updated or adapted [11, 10, 12]. Understanding the role of concept drift in effort estimation is therefore an important research challenge, with implications both for theory and for practice. To address this challenge, it becomes necessary to counteract drift: models that once produced reliable predictions may no longer provide useful results and must be retrained or adapted to the new data distribution [11, 13, 12]. Without such measures, estimation quality deteriorates, undermining the effectiveness of project planning.

This leads to two key research questions:

- Can concept drift be detected in the context of effort estimation?
- If so, can it be effectively counteracted using existing strategies?

1.1 Aim and scope of the thesis

The aim of this thesis is to investigate concept drift in task-level software effort estimation, with a particular focus on user stories and issue reports as they evolve over time in Agile projects. The research is guided by two central questions: (1) whether concept drift can be detected in task-level estimation datasets, and (2) whether it can be mitigated effectively using adaptive learning strategies. From these questions follow two hypotheses: first, that concept drift occurs in effort estimation datasets and reduces the accuracy of static predictive models; and second, that adaptive methods, such as drift detection and retraining, improve predictive performance.

The study uses the publicly available TAWOS dataset, with each project analyzed individually over time. The unit of analysis is the individual issue or user story, reflecting the task-level granularity common in Agile planning. A Random Forest Regression model serves as the baseline after preprocessing. Training begins with the oldest 10 percent of records and expands in 5 percent increments up to 80 percent, with models evaluated on the most recent 20 percent. A sliding-window evaluation illustrates how reliance on older data reduces accuracy. Concept drift is detected using the ADWIN algorithm, and models are subsequently retrained with weighted data to emphasize more recent information.

The scope of this thesis therefore lies at the intersection of machine learning for effort estimation and concept drift detection in evolving software projects. Unlike much of the prior work that has concentrated on project-level estimation methods, this study emphasizes the more fine-grained task level, where estimation is highly uncertain, more dynamic, and increasingly relevant in Agile practice. The work relies on publicly available datasets and academic literature from software engineering, machine learning, and drift detection. Some limitations are acknowledged, including the restricted detail of project metadata, the need for extensive preprocessing, and the computational demands of repeated model training and drift detection. Within these boundaries, the thesis provides an empirical study of concept drift in task-level effort estimation and evaluates the effectiveness of adaptive strategies in maintaining predictive accuracy over time.

1.2 Structure of the thesis

The structure of this thesis is as follows:

- Chapter 2 presents a review of the literature on effort estimation in IT projects, including the use of datasets such as TAWOS and Agile-oriented practices such as story points. It also discusses sliding-window strategies for handling temporal data, provides background on machine learning algorithms applied to effort estimation, and examines the problem of concept drift in data-driven systems, with particular attention to adaptive detection methods such as ADWIN.

- Chapter 3 describes the research methodology, outlining the goals of the study and the research questions it addresses. It presents the datasets used and their characteristics, details the preprocessing steps applied, and introduces the predictive models selected for analysis. The chapter also explains the research procedure step by step, defines the evaluation metrics used, and specifies the experimental framework for detecting drift and assessing adaptation strategies.

- Chapter 4 presents the results obtained from the research procedure. The outcomes are reported in the form of tables and supported by visualizations, providing a clear representation of the experimental findings and the observed behavior of the models.

- Chapter 5 discusses the results in relation to the two research questions formulated in this study. It interprets the findings, examines their implications, and reflects on possible explanations for the

observed outcomes. The chapter also addresses threats to validity, considering the limitations of the research design and the reliability of the conclusions.

- Chapter 6 provides the conclusions of the thesis. It restates the research goal and summarizes what has been accomplished. The chapter highlights the main observations and results obtained, offering a concise overview of the key contributions of the work.

Chapter 2

Background and related work

2.1 Effort estimation in IT projects

Effort estimation is one of the fundamental activities in software engineering and IT project management, forming the basis for planning, scheduling, and budgeting. Reliable estimates increase the likelihood of on-time and within-budget delivery, while inaccurate estimates have historically led to project overruns, quality deficits, and unmet stakeholder expectations [2]. From a management perspective, effort estimation is not only a technical challenge but also a strategic one, as it influences contract negotiation, resource allocation, and requirement sequencing.

The literature traditionally distinguishes between estimation at the *project level* and at the *iteration or task level*. Estimation at the project level has been the dominant subject of research for several decades and remains the foundation of most traditional software engineering cost estimation studies. Widely used approaches include expert-based methods such as Delphi estimation [1] and analogy-based reasoning where effort is inferred from similar past projects [14, 15], and algorithmic cost models such as COCOMO, or SLIM [1, 16]. Comprehensive reviews highlight that these project-level methods have been the main focus of the field for over forty years [17, 18]. Although these approaches provide more systematic estimates than informal judgments, their accuracy remains highly sensitive to organizational context, data quality, and evolving technological factors.

By contrast, estimation at the level of iterations, user stories, or individual tasks has received less attention in research, though it plays a central role in Agile methods. Agile teams typically estimate effort in terms of story points or velocity over short sprints using Planning Poker [1, 19]. Such estimates fluctuate frequently due to changing requirements and team dynamics. Task-level estimation concerns predicting the effort needed to complete small work items. While such tasks are smaller in scope and in principle easier to approximate, they remain challenging in practice due to cognitive biases, uncertainty in requirements, and variability in team experience [17]. This makes story-point and task-level estimation a relatively niche but increasingly relevant research area, particularly as Agile practices have become widespread in industry.

Despite many years of study, effort estimation at all levels remains inherently difficult. Uncertainty in requirements, rapidly changing technologies, differences in team capability, and well-established cognitive biases such as optimism bias or groupthink continue to undermine estimation accuracy. These challenges underline the importance of investigating more adaptive, data-driven approaches that can cope with evolving contexts and mitigate the effects of concept drift in software projects.

2.2 Machine learning in effort estimation

Machine learning has rapidly emerged as an increasingly valuable approach to software effort estimation that may provide features traditional models cannot. Unlike estimates based on expert judgment and estimates made using algorithmic methods, machine learning models leverage data from historical records directly, capturing very complex nonlinear relationships that exist between some combination of project attributes and level of effort required, as highlighted in the systematic review by Mahmood et al. (2021) [3].

Many machine learning methods have been used for effort estimation. Regression models can provide interpretability but often have difficulties combining non-linear dependencies, such as project size with development experience and team size. Decision trees show a higher capacity model than regression by splitting data into smaller regions that exhibit similar characteristics, which enables less flexible, yet more nuanced predictions. When fitted to complex data, ensemble models gain predictive performance by combining many weak learners to create a strong predictive model. Random Forest and Gradient Boosting are both ensemble models that have been utilized extensively in industry and academic research, with Random Forest being the most adopted approach due to its robustness against overfitting and ability to handle heterogeneous project attributes [4] .

In recent work, there have been efforts to exploit deep learning approaches, known as DeepSE (Deep Software Effort estimation). Neural networks can automatically extract features and learn representations, and have been shown to perform very well on large scale data sets. Deep learning models can automatically learn the complex dependencies in the historical data of the project, which may not be easily discoverable by human experts and simple algorithms. Although, deep learning methods often require a large training set and a great deal of computational power, which may hinder its implementation in many software engineering projects.

Machine learning methods have fundamental restrictions. The most elementary assumption made by most models is that the train and test dataset is from the same or similar underlying distribution. However, in the context of software projects, this assumption is unlikely to be applied. An important element of software projects is that, due to its dynamic nature, the underlying data distributions change gradually overtime. This process is known as concept drift and it undermines the long run effectiveness of machine learning models.

2.3 Concept drift

Machine learning models are typically formulated on the assumption the data distributions will not change over time. This assumption is rarely correct in practice, especially for dynamic environments, like project-oriented environments. As defined by Gama et al. (2014) [8], concept drift refers to changes in the statistical properties of data streams over time, typically expressed as shifts in the joint probability distribution, which may affect either the input features, the output labels, or their relationship. A model or models that were trained on historical data will not make accurate predictions on future data due to a drift occurring. Consequently, concept drift can be classified into several categories:

- Sudden drift -– occurs when the data distribution changes abruptly, for example when a team adopts a new technology overnight.
- Gradual drift — occurs when old and new concepts coexist for a period of time, such as when requirements evolve across several sprints.
- Incremental drift — arises from small but continuous changes that accumulate over time, for instance gradual improvements in team productivity.

- Recurring drift — takes place when previously observed patterns reappear, such as seasonal workload fluctuations.

Each of these forms of drift poses challenges for maintaining long-term predictive accuracy, particularly in effort estimation, where evolving requirements, shifting tools, and changing team composition are frequently encountered [9].

To address the issue of drift, several researchers have developed methods for detecting drift when it happens. Some methods, including the Drift Detection Method (DDM) or the Early Drift Detection Method (EDDM), monitor error rates and ring the alarm when there is a large jump in error rates. Other approaches include Adaptive Windowing (ADWIN) and a variety of statistical tests such as the Page–Hinkley test or the Kolmogorov–Smirnov test.

Adaptive Windowing (ADWIN) is one of the most widely adopted drift detection algorithms for data streams, introduced by Bifet and Gavalda in 2007 [11]. Its core principle is to maintain a variable-length window of the most recent observations and to continuously test whether the statistical distribution within this window has changed. At each time step, ADWIN splits the window into two sub-windows at every possible cut point and applies a statistical test (based on Hoeffding's inequality) to compare the means of the two segments. If the difference between the means is statistically significant, the older segment is discarded, indicating that concept drift has occurred and the distribution has shifted. If no significant difference is found, the window can grow to incorporate more historical data.

ADWIN relies on **Hoeffding's inequality** to bound the probability that the observed mean of a random variable deviates from its true expectation. Let $X_1, X_2, \ldots, X_n$ be independent random variables with values in $[0, 1]$, and let $\hat{\mu}$ be their empirical mean. Then, for any $\epsilon > 0$,

$$P(|\hat{\mu} - E[\hat{\mu}]| > \epsilon) \leq 2\exp\left(-2n\epsilon^2\right).$$

In ADWIN, the window is split into two sub-windows, W_0 and W_1, with respective means $\hat{\mu}_0$ and $\hat{\mu}_1$. A drift is detected if the difference between these means exceeds a threshold derived from Hoeffding's bound:

$$|\hat{\mu}_0 - \hat{\mu}_1| > \epsilon = \sqrt{\tfrac{1}{2m} \ln \tfrac{4}{\delta}},$$

where $m = \frac{1}{|W_0|} + \frac{1}{|W_1|}$ and δ is a confidence parameter (typically chosen to be very small).

This adaptive resizing gives ADWIN two important properties. First, it can react quickly to abrupt changes by shrinking the window immediately when large distribution shifts occur. Second, it provides stability during stationary periods by allowing the window to grow, thereby reducing false alarms and exploiting larger amounts of data. A notable advantage is that ADWIN is essentially parameter-free: unlike DDM or EDDM, which require the user to fix confidence levels or thresholds in advance, ADWIN automatically adjusts its sensitivity according to the statistical properties of the stream.

From a computational perspective, ADWIN (specifically, ADWIN2) has logarithmic amortized time and memory complexity with respect to the window size, which makes it efficient enough for real-time monitoring of large data streams. In the context of software effort estimation, these properties are particularly valuable because the extent and speed of change in issue trackers are unpredictable. Some projects exhibit gradual evolution of practices, while others undergo sudden shifts due to technology migration, changes in team composition, or altered estimation practices. ADWIN's ability to balance rapid adaptation with stability therefore makes it well-suited for this domain. In this thesis, ADWIN is used as the primary mechanism for drift detection, ensuring that the predictive models can adapt when the underlying distribution of issues and story points changes over time.

2.4 Related work in effort estimation and concept drift

Research on software effort estimation and concept drift spans several decades, with contributions ranging from classic algorithmic models to modern machine learning approaches. This section reviews selected works that are directly relevant to the scope of this thesis.

Moløkken-Østvold and Jørgensen (2003) [2] conducted a review of surveys on software effort estimation practices. Their study synthesized results from multiple industry surveys and revealed that inaccurate estimates remain a big problem in software development projects. They highlighted that a majority of projects suffer from effort overruns, often linked to optimism bias and insufficient consideration of risks. One of their main conclusions was that estimation errors are not merely technical issues but also organizational, influenced by human judgment and project context.

From their review, this thesis takes the recognition that inaccurate effort estimation is a persistent and fundamental challenge in software projects. Their findings, drawn from industry surveys, emphasize that estimation errors lead to frequent overruns and delays, and that uncertainty is an unavoidable characteristic of software development. This perspective supports the motivation for the present study, since it shows that estimation inaccuracy is not a new or marginal concern but one of the core issues in software engineering research.

In contrast, while their work attributes estimation errors largely to human judgment, optimism bias, and organizational influences, this thesis shifts attention toward systematic changes in the data itself. Instead of focusing on cognitive or managerial factors, the present research examines concept drift in issue-level datasets and explores adaptive machine learning strategies as a complementary explanation for declining model accuracy over time.

COCOMO II, which was introduced by Boehm et al. (2000) [1], is one of the prominent algorithmic models for estimating software costs and effort. COCOMO II formalized effort estimation as a mathematical function a set of characteristics of a project including size, complexity, reusability and capability of personnel involved. The model was developed to account for changes in software development including iterative development, and its reuse. COCOMO II is one of the foundational models in algorithmic estimation and requires calibration to relevant organizational data to improve the underlying estimate. COCOMO II has been commonly used and cited in research and taught in the academic context, yet it has been criticized for adaptability to the changing project environments in a shorter time frame compared than machine learning–based approaches.

From COCOMO II, this thesis takes the idea that effort estimation can be formalized into systematic models that rely on quantifiable attributes of software projects. COCOMO demonstrates that effort is not purely a matter of subjective judgment but can be expressed as a mathematical function of size, complexity, reuse, and personnel capability. This formalization underpins the rationale for using machine learning in this thesis, as both approaches share the premise that predictive models can improve estimation reliability if they capture the right features of project data.

In contrast, COCOMO II operates at the project level and assumes relatively stable relationships between input attributes and effort. It requires periodic manual recalibration to adapt to organizational contexts. By comparison, this thesis focuses on task-level data (user stories and issues) and challenges the assumption of stability by explicitly analyzing temporal evolution in data distributions. Rather than manual recalibration, the present work employs drift detection with ADWIN and retraining strategies to automatically adapt models as data characteristics change over time.

Mahmood et al. (2021) [3] presented a systematic literature review that examined multiple machine learning algorithms for estimating software effort. Their review included a wide range of studies, and they found strong evidence that machine learning models outperform traditional estimation techniques such as expert judgment and algorithmic models for software effort estimation. Their review emphasized

the applicability of models capable of handling complex nonlinear relationships in datasets, such as Random Forest and neural networks. At the same time, they highlighted limitations of machine learning estimation, including the need for large, high-quality datasets and the lack of generalizability across organizations.

From their systematic review, this thesis takes the evidence that machine learning algorithms, particularly ensembles such as Random Forests and neural networks, frequently outperform traditional approaches like expert judgment or algorithmic models. Their study demonstrates that ML techniques can capture complex, nonlinear relationships in effort data and are therefore better suited to the inherent uncertainty of software development. This validates the choice of Random Forest as a baseline model in the present thesis and supports the broader motivation for applying data-driven learning methods to effort estimation.

In contrast, their review primarily considers studies where models are trained and evaluated under static conditions, typically using random train/test splits. This thesis builds on their findings but extends them by considering the temporal dimension of data, investigating how model performance changes over time as concept drift occurs. Rather than benchmarking algorithms in isolation, the focus here is on how models degrade under drift and how adaptive strategies can restore accuracy, filling a gap in the long-term evaluation of ML-based effort estimation.

Rathore et al. (2022) [4] conducted another systematic literature review focusing on machine learning techniques for software effort estimation. Their review highlighted the extensive adoption of ensemble learning methods such as Random Forest and Gradient Boosting, which were consistently among the best-performing models. They also identified trends in the use of deep learning, although results were mixed due to data scarcity in software engineering. The authors concluded that while machine learning methods improve estimation accuracy in many contexts, the lack of standardized datasets and evaluation protocols limits comparability across studies.

From their review, this thesis takes the insight that ensemble methods, especially Random Forest and Gradient Boosting, are consistently among the most effective algorithms for software effort estimation. Their findings strengthen the case for using Random Forest as the primary baseline in this study and justify focusing on ensemble learners when exploring adaptive retraining strategies. In addition, their identification of deep learning's limited reliability due to data scarcity reflects the practical constraints that also shape the dataset choice in this thesis.

In contrast, their work aggregates results across many studies and focuses on comparing algorithmic families in terms of average performance. By comparison, this thesis narrows the scope to a single dataset, the TAWOS corpus, and investigates performance under temporal dynamics rather than across algorithm classes. The contribution lies in showing that even strong ensemble models like Random Forests are not immune to performance degradation when data distributions drift, and in testing how drift detection and retraining can help maintain their accuracy.

Gama et al. (2014) [8] published a comprehensive survey on concept drift adaptation, establishing a framework for understanding drift in data-driven systems. They defined concept drift as changes in the joint probability distribution of inputs and outputs, distinguishing between real drift and virtual drift. The survey categorized drift into types such as sudden, gradual, incremental, and recurring. They reviewed detection methods, including error monitoring and statistical tests, as well as adaptation strategies like sliding windows, ensembles, and retraining. Their survey remains one of the most authoritative references on drift and its handling.

From their survey, this thesis takes the conceptual framework of concept drift and the taxonomy of drift types (sudden, gradual, incremental, recurring). Their discussion of detection techniques, such as error monitoring and statistical hypothesis testing, and adaptation strategies like sliding windows and ensembles, provides a theoretical foundation for applying drift methods in other domains. This thesis

draws directly on that foundation by treating effort estimation as a streaming-like problem in which issue-level data arrive sequentially and distributions evolve.

In contrast, Gama et al. present a broad and domain-general overview, focusing on generic data streams such as finance or sensor data. They do not address software engineering or effort estimation as an application area. This thesis extends their framework into a new domain by empirically studying drift in software project data and operationalizing drift detection with ADWIN in the context of effort prediction, where it has not been previously applied.

Žliobaitė (2016) [9] provided an overview of concept drift applications across various domains, including finance, medicine, and engineering. The study emphasized that concept drift is a ubiquitous problem in predictive modeling wherever data distributions evolve over time. The paper highlighted challenges in maintaining long-term model accuracy in environments with high variability. For software projects, this is particularly relevant, as requirements, tools, and team composition change frequently. Žliobaitė concluded that adaptive methods are essential in any setting where stability cannot be assumed, providing further motivation for studying drift in effort estimation.

From her overview, this thesis takes the recognition that concept drift is a ubiquitous problem across predictive modeling domains, arising whenever the relationship between inputs and outputs changes over time. The emphasis on long-term model sustainability and the challenges of adapting to highly variable environments resonates strongly with the problem faced in software projects, where requirements, tools, and teams often change. This provides additional justification for treating effort estimation as a drift-prone prediction task.

In contrast, Žliobaitė's survey remains broad and illustrative across domains like finance, medicine, and engineering, without focusing on software development. This thesis narrows the lens to task-level effort estimation in Agile projects, providing empirical evidence that drift is not only a theoretical possibility but a practical reality in this setting. By demonstrating drift and adaptation in the TAWOS dataset, it validates her claim within a concrete domain where adaptive modeling has direct project-management implications.

Adaptive Windowing (ADWIN) was presented by Bifet and Gavaldà (2007) [11] as a drift detection method developed for data streams. The way ADWIN works is by maintaining a sliding window of previous observations and dividing it into a number of subwindows to determine if their average differs significantly. It does not require any manual parameter tuning. In the experiments in their paper, they conclude that ADWIN provides a proper balance between sensitivity to change and stability in stationary settings, and is a suitable option for certain adaptive learning systems.

From their introduction of ADWIN, this thesis takes a practical algorithm for drift detection that requires no manual parameter tuning and provides a balance between sensitivity to change and stability when no drift occurs. ADWIN's statistical guarantees based on Hoeffding's inequality make it suitable for detecting change points in evolving data streams, and these properties motivate its selection as the main detection mechanism in this thesis.

In contrast, Bifet and Gavalda validate ADWIN on synthetic and generic data streams, without reference to software engineering tasks. This thesis extends its application into the domain of effort estimation, testing whether ADWIN's detection of distributional shifts improves the predictive accuracy of models trained on evolving task-level datasets. The contribution is not the algorithm itself but its novel application to this problem domain.

Minku and Yao (2012) [20] investigated whether cross-company data could improve software effort estimation. They conducted experiments using multiple datasets from different organizations to test whether models trained on external data could outperform within-company models. Their results showed that cross-company estimation often performed poorly due to differences in organizational practices and contexts. However, they also identified scenarios where external data could help, particularly when

combined with appropriate adaptation techniques. This work highlighted the challenges of generalizing estimation models across organizational boundaries and motivated further work on transfer learning in effort estimation.

From their study, this thesis takes the insight that adaptation is necessary when estimation models are transferred across contexts. Their findings that cross-company estimation often underperforms without adaptation reinforce the general principle that distributional differences reduce model accuracy, which is directly relevant for this thesis.

In contrast, they focus on cross-organizational transferability and how models generalize between companies. This thesis instead investigates temporal transferability within a single dataset, showing that even in one organizational context, accuracy declines as distributions drift over time. By focusing on within-project temporal dynamics, it addresses a different but complementary dimension of the transferability problem.

In a follow-up study, Minku and Yao (2013) [21] explored the use of ensembles and locality to improve software effort estimation. Their approach involved combining multiple models and giving more weight to models trained on data similar to the target project. Results demonstrated that ensemble methods could significantly improve predictive accuracy compared to single models, particularly in heterogeneous datasets.

From their ensemble approach, this thesis takes the idea that weighting models based on data similarity can improve predictive performance. Their results confirm that ensembles can outperform single models in heterogeneous datasets, which informs the use of Random Forest in this thesis.

In contrast, their approach focuses on locality in terms of project similarity, giving more weight to models trained on data from comparable projects. This thesis instead focuses on temporal locality, prioritizing recent data within the same project and using ADWIN to decide when to update models. Thus, while both works leverage ensemble logic, the adaptation mechanism differs.

Lokan and Mendes (2009) [13] applied a moving-window approach to software effort estimation. They argued that as project data evolve over time, models should rely more heavily on recent observations rather than all historical data. Their experiments demonstrated that moving-window models can better adapt to changes in productivity and project characteristics, improving estimation accuracy compared to static approaches. This work was an early application of temporal adaptation techniques to effort estimation and directly connects to the concept of drift handling.

From their moving-window approach, this thesis takes the idea that recent data may be more relevant for prediction than older data. Their experiments show that static models trained on all historical data risk becoming obsolete as productivity and project characteristics evolve, which strongly motivates the adaptive methodology in this thesis.

In contrast, they rely on heuristic windowing without explicit drift detection, applying the intuition that "recent is better" without formal change detection criteria. This thesis builds on their intuition but extends it with drift detection (ADWIN) and retraining strategies, formalizing the conditions under which older data should be discarded.

Tawosi, Al-Subaihin, and Sarro (2022) [6] studied the effectiveness of clustering for story point estimation in Agile projects. They investigated whether grouping similar user stories improved prediction of effort in terms of story points. Their results indicated that clustering-based methods could enhance accuracy, particularly when datasets included large variability in story descriptions.

From their clustering-based approach, this thesis takes the idea that task-level user stories are a suitable unit of analysis for machine learning–based estimation. Their results demonstrate that refining story representations can improve predictions, which supports the decision in this thesis to work with issue-level data.

In contrast, their method focuses on clustering stories to improve representation, whereas this thesis focuses on temporal dynamics in story data and investigates drift detection and adaptive retraining as a way to preserve accuracy over time.

In another study, Tawosi, Moussa, and Sarro (2022) [7] examined the relationship between story points and actual development effort in Agile open-source projects. They analyzed large-scale datasets to test whether story points provide a reliable basis for estimating effort. The results showed that the correlation between story points and actual effort is often weak and inconsistent across projects. Their findings confirmed that story point estimation, while widely used in Agile practice, remains problematic as a basis for accurate prediction. This highlights the persistence of estimation challenges in Agile contexts.

From their study on story points, this thesis takes the finding that story points correlate inconsistently with actual effort, highlighting the challenges of task-level estimation in Agile contexts. This motivates further investigation into why prediction accuracy declines over time.

In contrast, their work attributes the weak predictive power to the limitations of story points themselves. This thesis extends their perspective by showing that concept drift contributes further to the unreliability of story points, and that adaptive modeling strategies can partially mitigate this degradation.

Chapter 3

Research method

3.1 Goals and questions

The overarching objective of this thesis is to investigate the role of concept drift in software effort estimation for IT projects. In particular, the study aims to examine whether concept drift occurs in this context and how it influences the reliability of predictive models. It also investigates which strategies can be applied to detect and counteract drift in order to improve the accuracy and adaptability of estimation methods. Building on these objectives, the research is guided by two main questions:

RQ1: Does concept drift occur in the context of effort estimation for IT projects?

This question is critical because the existence of drift determine whether adaptive methods need to be used. If project data is relatively constant, static models would likely be sufficient. But if concept drift is present and constant review of ability to estimate is ignored, then systematically worsening estimates will still result. Therefore, understanding and defining concept drift is necessary before constructing a sound estimation framework.

RQ2: If concept drift occurs, can it be effectively counteracted using existing strategies such as drift detection and model retraining?

This question builds on the first question, and investigates not only whether drift is occurring, but also whether the drift can be managed in a practical manner. Several methods identified in the literature, such as adaptive windowing and retraining, show potential for addressing effort estimation. However, none have yet been systematically studied or applied in this particular modeling area. If effectively implemented, such methods could enhance planning reliability, reduce the risks of under- and overestimation, and promote adaptive, data-driven practices in software engineering.

3.2 The TAWOS dataset

The experiments in this thesis are based on the TAWOS dataset (Task and Work in Open Source) [22], which contains 458,232 issues from 39 open-source projects drawn from 12 public Jira repositories. It has recently been introduced to support research on Agile effort estimation and story point prediction. The dataset was created by Tawosi and colleagues and was derived from issue-tracking systems of large open-source projects. It represents one of the few publicly available datasets that provide project-level information with sufficient temporal detail to allow for the investigation of concept drift in effort estimation. Table 3.1 below summarizes the 15 projects retained for this study after data preprocessing, providing a short description of each project together with the number of usable issues.

Table 3.1: Projects retained from the TAWOS dataset after preprocessing, with descriptions and number of issues.

ID	Key	Name	Description	Issues
1	XD	Spring XD	Data ingestion and real-time analytics platform in the Spring ecosystem	3372
3	NEXUS	Sonatype Nexus	Artifact repository manager for software builds and dependencies	1715
4	MESOS	Apache Mesos	Cluster manager for scalable distributed systems (e.g., Hadoop, Spark)	3083
5	USERGRID	Apache Usergrid	Backend-as-a-Service (BaaS) for web and mobile apps (retired 2021)	464
8	APSTUD	Appcelerator C++ Driver	Component of Appcelerator ecosystem providing language/tooling support	696
11	TIDOC	Appcelerator Titanium Docs	Documentation platform for the Titanium mobile SDK	1203
12	TIMOB	Appcelerator Titanium SDK	Cross-platform mobile app development framework in JavaScript	4061
13	TISTUD	Appcelerator Studio	IDE for cross-platform mobile development using Titanium	2889
24	FAB	Hyperledger Fabric	Modular blockchain framework for permissioned distributed ledgers	533
25	INDY	Hyperledger Indy	Distributed ledger for decentralized identity management	621
28	DM	LSST Data Management	Infrastructure for handling large-scale astronomical data from the LSST project	17629
29	DURACLOUD	Lyrasis DuraCloud	Cloud-based digital preservation and storage service	634
34	MDL	Moodle	Open-source learning management system (LMS) for online education	1131
36	MULE	Mulesoft Mule	Lightweight enterprise service bus and integration platform	2716
43	EVG	MongoDB Evergreen	Continuous integration and testing framework for MongoDB builds	4706

Together, these projects represent a cross-section of contemporary open-source software engineering, spanning distributed systems, cloud infrastructure, blockchain, developer tools, mobile frameworks, learning management, and data preservation. By including projects of varied size and domain, the selected subset captures the heterogeneity of real-world Agile software development. At the same time, the common structure of the dataset ensures that the analyses conducted in this thesis remain comparable across domains.

Each original observation of the dataset corresponds to a work item – usually a user story, feature request or bug report – that was assigned story points and was closed over time. Accordingly, the dataset includes planning information (i.e., story points, issue type, priority), and outcome information (i.e., resolution time, completion status). All projects contain several tens of thousands of issues. However, projects vary widely in scale from a few hundred issues to several thousand issues.

The TAWOS dataset provides real-world Agile data at the level of user stories and tasks, where estimation is particularly challenging. It uses temporal order, so it allows examination of how models perform over time, which is conducive for studying concept drift. While there are strengths to the dataset, there are also limitations. It is from open-source projects only and will likely not represent the practices of industrial software development entirely. Also, the textual descriptions are inconsistent in quality and completeness - issues are not completed to any description standard due to the informal nature of issue tracking.

3.3 Data preprocessing

The raw TAWOS dataset consisted of a single, very large CSV file containing issue reports from multiple projects. In its unprocessed state, the dataset was not suitable for analysis, both because of its size and because of inconsistencies, missing values, and redundant attributes. To prepare the data for experiments, a multi-stage preprocessing pipeline was implemented. All steps were automated with Python scripts to ensure reproducibility and efficiency.

The first challenge was the size of the input file. Loading the entire dataset into memory was not feasible on standard hardware. To overcome this, the file was divided into smaller slices using a streaming CSV reader. Each slice was defined by a starting line number and a fixed number of rows, with the header row always preserved. Rows that could not be parsed due to delimiter or encoding errors were skipped. This procedure made it possible to process large volumes of data in a controlled and memory-efficient manner.

Following the splitting of the dataset, the next step was cleaning and reducing attributes. The raw issue reports were first normalized to a common image, CSV format for compatibility with the required analysis tools. Several common fields (e.g., identifiers, URLs, and status information) were deemed irrelevant during the study and were removed. The remaining variables were normalized by converting dates into a common datetime format and converting numeric variables such as story points and effort minutes to numeric types, discarding invalid records. In addition, records missing a valid project identifier were excluded since they could not be linked reliably to a project.

At this stage, the data was routed into individual files by project. All issues were grouped based on their “Project_ID”, and all issues belonging to the same project were stored in one CSV file named after the project identifier. The preprocessing also included filtering of the target variable. For the story point–based experiments, only issues with valid story point values between 1 and 19 were kept. This restriction removed issues with missing values, non-numeric entries, or extreme outliers that could distort model training. Issues lacking a valid creation date were excluded, as preserving temporal integrity was crucial for chronological analyses.

Finally, all project-level datasets were chronologically sorted by creation date, ensuring that the temporal order of issues was preserved. As a last filtering criterion, only projects with at least 450 usable issues after cleaning were retained. This yielded a final corpus of 15 projects, each represented as a clean, and chronologically ordered CSV file. These project datasets provided a reliable and consistent basis for all subsequent experimental work presented in this thesis.

3.4 Baseline model

The implementation was carried out in *Python 3.11.5*, using libraries including *pandas*, *NumPy*, *scikit-learn*, and *sentence-transformers*. All subsequent experimental procedures were based on a common baseline setup combining feature engineering and the same machine learning model. Categorical attributes (“Type” and “Priority”) were encoded using *One Hot Encoding*, with encoders fitted on the

full dataset of each project to ensure consistency between training and testing. The output was numerical embeddings that reflected the meaning of the issues, allowing the model to make use of the text information that could not be captured in its original form.

The resulting feature vectors consisted of the text embeddings together with the categorical encodings. The target variable ("Result") was log-transformed using *log1p* to reduce skewness and stabilize variance. The baseline model used in all experiments was a Random Forest regressor, implemented using scikit-learn library. It was chosen for its robustness to overfitting and strong performance in prior effort estimation studies. A validation wrapper automated hyperparameter tuning and metric reporting. The best results were obtained with a Random Forest configured with 100 estimators and no maximum depth. This produced consistent datasets ready for temporal analysis, sliding windows, and drift detection.

3.5 Procedure

3.5.1 Baseline evaluation using temporal folds

Before introducing concept drift detection, it was essential to establish a baseline evaluation framework. Unlike random cross-validation, which mixes records without regard to their ordering, temporal folds preserve the chronological sequence of issues. This ensures that training data always procedes testing data in time. It better reflects the real conditions of project planning where future information is not available at the time of estimation.

In the implementation, each project was processed independently. Training began with the oldest 10% of issues, after which the training set was incrementally expanded in 5% steps up to 80% of the project history. At every step, the most recent 20% of issues served as a fixed test set. The fragment of code below illustrates this procedure.

```
test_X, test_y = prepare_latest_data(full_df, percent=20,
                 type_ohe=type_ohe, priority_ohe=priority_ohe)

all_results = []

for percent in percentages:
    train_X, train_y = prepare_oldest_data(full_df, percent,
                       type_ohe=type_ohe, priority_ohe=priority_ohe)

    base_model = RandomForestRegressor(random_state=42)
    validator = Validator(base_model)
    best_model, metrics = validator.tune_and_validate(train_X, train_y,
                          test_X, test_y, param_grid)
```

This design made it possible to examine how models trained on different portions of the past performed on unseen future data. It avoided information leakage that would occur if future issues were included in training, and it ensured that the evaluation framework was consistent across all projects.

3.5.2 Sliding window analysis

In addition to temporal folds, a sliding window procedure was implemented to further investigate how models perform when trained on consecutive segments of project data. This experiment evaluates short-horizon generalization by training on a contiguous block of recent history and testing on the immediately following block.

For each project, the script loads the dataset and skips files with fewer than 24 rows. Each project is partitioned into 12 equal-sized contiguous chunks. The procedure iterates over consecutive pairs: for window *i*, the model is trained on chunk *i* and tested on chunk *i+1*. For every window, a Random Forest regressor is initialized with a fixed parameter grid (*nestimators=100*, *maxdepth=None*) and trained using sample weights proportional to the back-transformed target values normalized by their maximum. Within this small grid, the configuration that achieves the lowest Mean Absolute Error (MAE) on the current test chunk is selected by a lightweight validation step.

```
chunks = [df.iloc[i * size: (i + 1) * size].reset_index(drop=True) for i in range(num)]

all_results = []

for i in range(num - 1):

    df_train = chunks[i]
    df_test = chunks[i + 1]

    X_train, y_train = prepare_data_from_df(df_train, model, type_ohe, priority_ohe)
    X_test, y_test = prepare_data_from_df(df_test, model, type_ohe, priority_ohe)

    base_model = RandomForestRegressor(random_state=42)
    validator = Validator(base_model)
    best_model, metrics = validator.tune_and_validate(X_train, y_train, X_test,
                          y_test, param_grid)

    all_results.append({"window": i + 1,})
```

This design reflects a scenario where estimation relies only on the most recent information available, rather than the entire project history. It allows the experiments to examine how well models adapt when older data is discarded and only recent context is used for prediction. The sliding window approach therefore complements the temporal folds by providing an alternative view of model behavior in evolving project environments, focusing on the short-term predictive value of the most recent data.

3.5.3 Concept drift detection and adaptive retraining

A central part of this thesis is the study of concept drift in software effort estimation and the development of strategies to detect and adapt to such drift. Static models, trained once on a fixed historical dataset, assume that the underlying distribution of project data remains stable over time. In practice, software projects evolve: requirements change, technologies are replaced, and teams adapt their practices. These changes can shift the statistical properties of the data, causing models trained on past information to lose predictive accuracy. To address this problem, two complementary implementations were developed: drift detection without adaptation and drift detection with adaptive retraining.

In the first setup, the Random Forest model is trained once on an initial prefix of 100 issues and remains unchanged for the remainder of the project. Predictions are made sequentially, and the per-issue relative error is monitored by the ADWIN algorithm with parameter $\delta = 0.01$ [23]. ADWIN maintains a dynamic window of error values and signals drift when the mean of the most recent segment differs significantly from the older segment. The implementation below shows when a drift is detected, the procedure records the drift index and computes two error summaries: a short pre-drift window (20 issues

before the detection point) and a short post-drift window (20 issues after the detection point, without retraining). These metrics provide a baseline view of how error evolves around detected drift events when no adaptation is applied.

```
if adwin.drift_detected:
    drift_idx = i - INITIAL_TRAIN_SIZE
    drift_points.append(drift_idx)

    before = errors[-BEFORE_WINDOW:] if len(errors) >= BEFORE_WINDOW else errors

    after_errors = []
    for j in range(i + 1, min(i + 1 + AFTER_WINDOW, len(X_all))):
        x_j = X_all[j].reshape(1, -1)
        y_j_true = y_all[j]
        y_j_pred = model.predict(x_j)[0]
        err_j = abs(np.expm1(y_j_true) - np.expm1(y_j_pred)) / np.expm1(y_j_true)
        after_errors.append(err_j)
```

The second setup extends the baseline by introducing an adaptation mechanism that couples drift detection with model retraining. When ADWIN signals drift, the Random Forest is retrained on the most recent window of data ending at the detection point (up to 100 issues, corresponding to the initial training size). To prevent over-reliance on limited recent data and to smooth the transition between contexts, the system also maintains snapshots of older models. After retraining, predictions are generated as a convex combination of the current model and the ensemble of old models, with a fixed weighting scheme: 30% weight assigned to the predictions of old models and 70% to the newly trained model. This blending ensures that older knowledge is not discarded entirely, while still prioritizing the adapted model. As in the baseline case, pre-drift and post-drift error windows of 20 issues are recorded for each detected drift.

Together, these two approaches serve complementary purposes. The no-retraining version provides a baseline characterization of error behavior around drift points, showing the unmitigated effects of evolving data distributions. The retraining version demonstrates the potential benefits of adaptive strategies, with a specific policy combining recent-window retraining and weighted blending of past and current models. Both implementations use the same detection protocol and evaluation setup, enabling clear analysis of the two guiding research questions: whether concept drift can be detected in effort estimation, and whether adaptive retraining can mitigate its effects.

3.6 Metrics

The evaluation of predictive performance in this thesis relies on a combination of global accuracy measures and localized drift-sensitive measures. This dual approach was chosen to capture both overall model quality across entire datasets and short-term behavior around points of distributional change. Each metric provides a different perspective, and together they form a comprehensive basis for assessing predictive models in the context of software effort estimation and concept drift.

The first group of metrics evaluates predictive accuracy in a global sense, summarizing performance across all available data. Four complementary measures were employed. The *Mean Absolute Error (MAE)* represents the average magnitude of prediction errors. It is computed as the mean of absolute differences between predicted and actual effort values. The MAE is defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{3.1}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

Unlike squared-error measures, it treats large and small deviations proportionally, making it less sensitive to outliers.

The *Mean Squared Error (MSE)*, by contrast, squares the differences between predicted and observed values before averaging. The MSE is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.2}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

This property penalizes large deviations more heavily than small ones, which makes MSE sensitive to extreme cases where the model performs poorly. MSE is widely used in regression contexts because of its mathematical convenience and its ability to highlight substantial errors.

The *Coefficient of Determination (R^2)* provides a relative measure of model fit by quantifying the proportion of variance in the observed values that is explained by the predictions. The R^2 is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{3.3}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value, $\bar{y}$ is the mean of observed values, and n is the number of observations.

A value of 1 indicates perfect explanatory power, a value of 0 indicates equivalence with using the mean of observed values as a predictor, and negative values indicate performance worse than this simple baseline. Unlike absolute error measures, R^2 conveys explanatory capacity in relation to variance, which is particularly useful when comparing predictive models across different datasets or experimental conditions.

The *Mean Magnitude of Relative Error (MMRE)* complements these measures by focusing on relative rather than absolute errors. It is defined as the average of the absolute error normalized by the actual observed value, thereby producing a dimensionless score. The MMRE is defined as:

$$MMRE = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i} \tag{3.4}$$

where y_i is the observed value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

This property makes MMRE suitable for comparing models across datasets of different scales. However, it is also known to be sensitive to small denominators, where relatively minor deviations can inflate the relative error. Despite this limitation, MMRE remains widely reported in software effort estimation literature, which makes it valuable for comparability with prior studies.

Localized metrics were introduced to analyze model performance around detected concept drift events. These measures are particularly relevant in adaptive learning settings, where understanding how errors evolve before and after a distributional shift is essential.

The *Mean Error Before Drift* is defined as the average relative error across a fixed window of 20 predictions immediately preceding a drift event. This metric captures whether error rates were already increasing in the lead-up to drift detection, thereby reflecting the model's stability in the pre-drift period. The Mean Error Before Drift is defined as:

$$E_{\text{before}} = \frac{1}{k} \sum_{j=i-k}^{i-1} \frac{|y_j - \hat{y}_j|}{y_j} \tag{3.5}$$

where y_j is the observed value at time j, $\hat{y}_j$ is the prediction at time j, k is the window size, and i marks the drift event.

In parallel, the *Mean Error After Drift* measures the average relative error over the 20 predictions following a drift event. This metric highlights the immediate consequences of drift, offering insight into whether prediction quality deteriorates sharply or remains stable once the data distribution changes. The Mean Error After Drift is defined as:

$$E_{\text{after}} = \frac{1}{k} \sum_{j=i+1}^{i+k} \frac{|y_j - \hat{y}_j|}{y_j} \tag{3.6}$$

where y_j is the observed value at time j, $\hat{y}_j$ is the prediction at time j, k is the window size, and i marks the drift event.

Finally, it is important to note that since the target variable was log-transformed during training to reduce skewness, all performance metrics were computed after back-transforming predictions to the original scale. This ensures that reported values remain interpretable in terms of actual effort and are directly comparable to prior research in the domain.

Chapter 4

Results

In order to keep the presentation of results clear and concise, a consistent subset of five representative projects was selected for all experiments. These projects were chosen as they capture a range of behaviors observed in the dataset and provide a balanced view of model performance. In some cases, additional projects are also included where they demonstrated either stronger results or highlighted particularly noteworthy aspects of the evaluation. This approach was necessary since the complete set of experiments involves 15 projects evaluated usually across four different metrics, which would be impractical to present in full within the chapter. For completeness and transparency, the full collection of results, including all projects and metrics, is made available in the *GitHub* repository at github.com/julia-mula/concept-drift.

4.1 Temporal folds

Table 4.1 presents the results of the temporal fold evaluation, where different proportions of historical data (10%, 35%, 50%, 75%, and 80%) were used for training. The table reports performance for four projects using four evaluation metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), the coefficient of determination (R^2), and the Mean Magnitude of Relative Error (MMRE). For each project, the best performance for a given metric across the folds is highlighted in bold for ease of comparison. Overall, the table provides a detailed view of how model performance changes as the percentage of historical data used for training is varied.

Table 4.1 shows the results of the temporal fold evaluation for five representative projects. Overall, MAE and MSE values remain relatively stable across folds but tend to improve when more historical data is included, with the lowest errors most often observed at the 75% fold. For several projects, such as Project 12 and Project 36, this fold also marks the point where R^2 values improve noticeably, shifting from negative to positive in the case of Project 12 and reaching the highest value in Project 36. In contrast, R^2 values for Project 1 and Project 13 remain low or negative across all folds, showing limited variance explanation. MMRE values fluctuate more irregularly, with best results appearing at different folds for different projects (e.g., 35% in Project 3 and 75% in Project 36). These patterns highlight that while adding more data often leads to lower error, the benefit is not uniform across all projects and metrics.

TABLE 4.1: Comparison of metrics' values across different percentages of historical data.

Project ID	**Metric**	10%	35%	50%	75%	80%
1	MAE	1.6365	1.6187	1.6261	**1.6024**	1.6039
	MSE	4.8086	4.4911	4.3531	**4.1217**	4.1469
	R^2	0.0176	0.0824	0.1106	**0.1579**	0.1528
	MMRE	0.6645	0.7100	**0.7475**	0.7436	0.7308
3	MAE	0.9534	0.9883	1.1020	**1.0292**	0.9883
	MSE	2.1939	2.3746	2.9111	**2.6108**	2.3980
	R^2	-0.0916	-0.1815	-0.4485	**-0.2990**	-0.1931
	MMRE	**0.5187**	0.4504	0.4172	0.3990	0.4121
12	MAE	3.0394	2.4655	2.4421	**2.4052**	2.4458
	MSE	17.9113	11.5640	11.4865	**11.3879**	11.6256
	R^2	-0.5166	0.0208	0.0274	**0.0358**	0.0156
	MMRE	0.5842	**0.6840**	0.6647	0.6259	0.6412
13	MAE	1.8076	1.7470	1.6811	**1.6724**	1.6690
	MSE	6.8787	6.0277	5.8856	5.8458	**5.7106**
	R^2	-0.1679	-0.0234	0.0007	0.0075	**0.0304**
	MMRE	0.4490	**0.4744**	0.4505	0.4386	0.4347
36	MAE	2.0718	2.0902	2.0203	**1.9355**	1.9650
	MSE	7.7698	7.6114	7.3352	**6.9374**	7.1952
	R^2	0.0654	0.0845	0.1177	**0.1656**	0.1345
	MMRE	0.9465	0.9459	0.9112	**0.8465**	0.8476

In Figure 4.1, we can see the metrics calculated for Project 13. MAE and MSE generally decrease as the percentage of training data increases, with the lowest errors appearing toward the higher folds. The R^2 values improve steadily, moving from negative in the early folds to slightly positive in the later ones, indicating a gradual gain in explanatory power. MMRE fluctuates across folds without a consistent downward trend, but overall this project is an example of clear improvement over time when more recent historical data is used.

In comparison, Figure 4.2 shows unstable performance across metrics in Project 3. MAE and MSE values fluctuate across the folds, with both metrics peaking in the middle ranges before decreasing again toward the higher percentages of data. The R^2 values remain negative throughout but vary considerably, showing the weakest performance in the mid-range folds. MMRE decreases more consistently, with lower values appearing in the later folds. Overall, this project illustrates not a steady performance across metrics, but still shows some improvement in error reduction when more recent historical data is included.

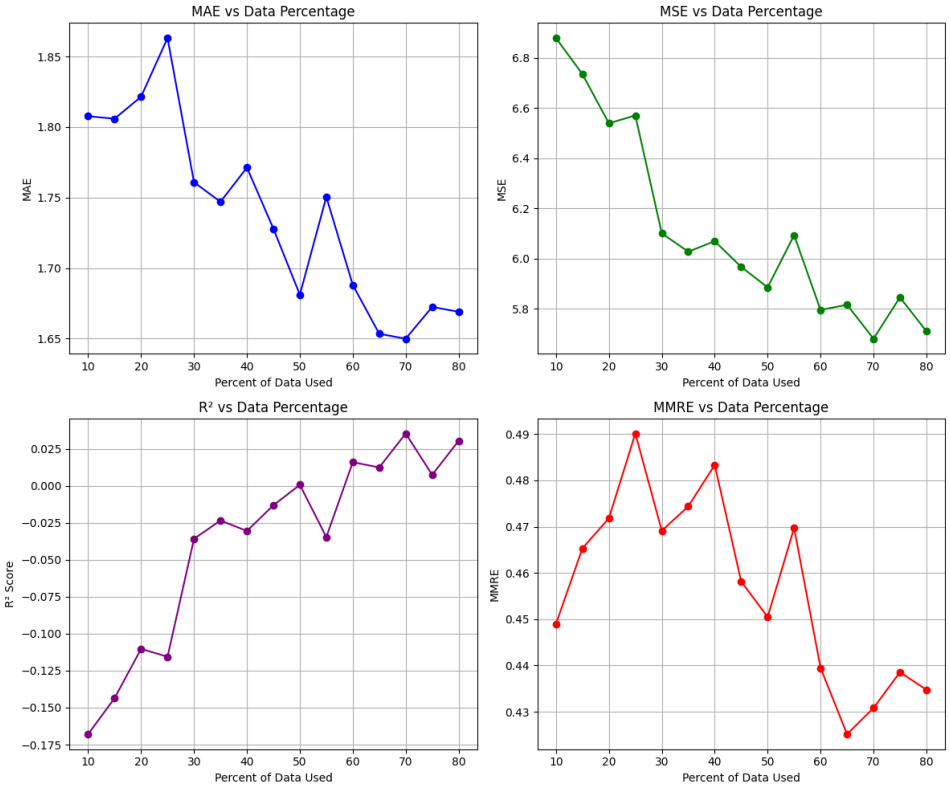


FIGURE 4.1: Performance metrics across different percentages of historical data in Project 13.

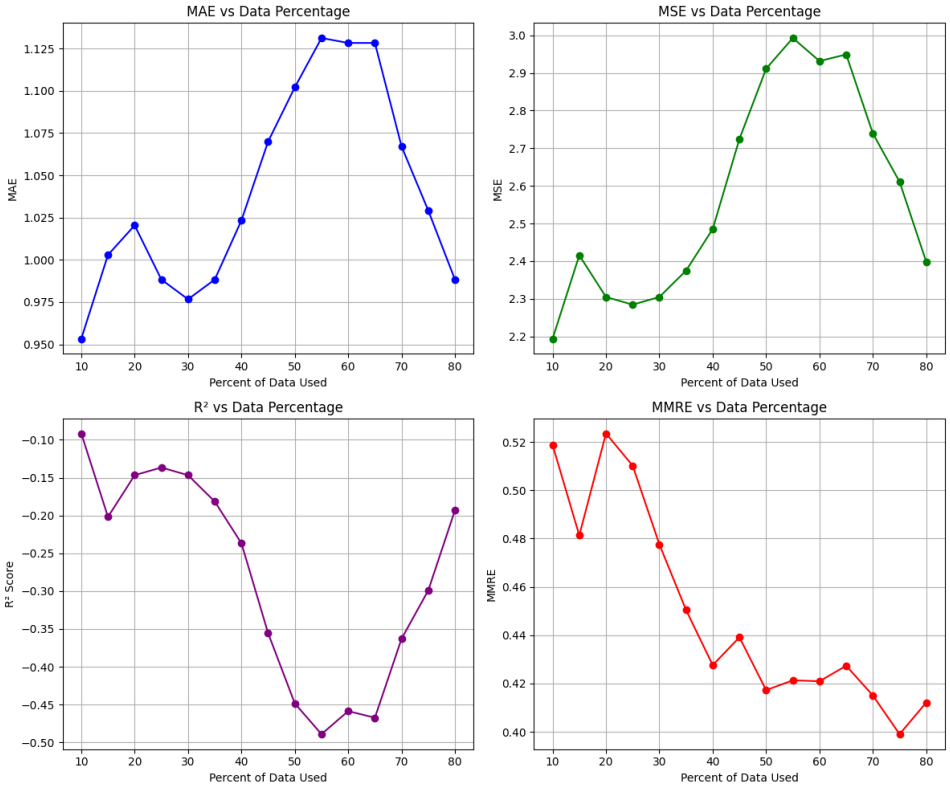


FIGURE 4.2: Performance metrics across different percentages of historical data in Project 3.

4.2 Sliding window

Table 4.2 presents the results of the sliding window evaluation for five representative projects. Performance is reported using four metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), the coefficient of determination (R^2), and the Mean Magnitude of Relative Error (MMRE). Results are shown for five selected windows (W1, W3, W5, W8, W11), and the best performance for each metric within a project is highlighted in bold.

Across projects, the lowest error values (MAE and MSE) are most often observed in later windows, particularly W5, W8, and W11. Project 12 shows a clear improvement, with MAE decreasing from 3.72 in W1 to 1.89 in W8 and MSE dropping from 25.53 to 8.82, while Project 36 follows a similar trend, reaching its lowest MAE and MSE at W8. In terms of R^2, middle and later windows also show better performance, with Project 12 moving from negative values in early windows to positive at W8 and Project 36 achieving its highest R^2 in the same window, whereas Project 3 remains negative across all windows despite competitive MAE values. MMRE varies by project but typically reaches its best values between W5 and W11, such as Project 13 with 0.35 at W5, Project 1 with 0.57 at W5, and Project 36 with 0.68 at W8, the same window that also produced its lowest error values.

Together, these results indicate that performance in the sliding window setting changes noticeably depending on the position of the training and testing window. While some projects achieve strong results in earlier windows, the majority show improved performance in middle or later windows, with W5 and W8 appearing most often as points of best performance across multiple metrics.

Table 4.2: Comparison of metrics across selected sliding windows (W1, W3, W5, W8, W11) for representative projects.

Project ID	**Metric**	W1	W3	W5	W8	W11
1	MAE	1.33	1.77	1.92	1.83	**1.48**
	MSE	2.90	6.37	7.72	5.11	**3.61**
	R^2	0.09	0.05	0.05	**0.13**	0.10
	MMRE	0.79	0.57	0.74	**0.82**	0.74
3	MAE	0.78	1.07	**0.44**	0.97	0.94
	MSE	1.23	2.12	**0.39**	2.22	2.42
	R^2	-0.10	-0.07	-0.15	-0.15	**-0.03**
	MMRE	0.71	1.07	0.65	**0.51**	0.53
12	MAE	3.72	2.50	2.23	**1.89**	1.98
	MSE	25.53	11.85	9.57	**6.88**	8.01
	R^2	-0.46	-0.05	0.05	0.02	**0.00**
	MMRE	0.66	0.63	0.74	0.66	**0.52**
13	MAE	2.98	2.02	**1.39**	1.40	2.00
	MSE	15.21	7.10	**3.55**	3.59	7.51
	R^2	-0.78	0.02	**0.09**	0.07	-0.08
	MMRE	0.42	0.47	**0.35**	0.35	0.52
36	MAE	2.41	2.50	2.58	**2.24**	2.10
	MSE	9.84	12.29	13.19	**11.15**	10.93
	R^2	-0.06	-0.02	0.11	**0.12**	-0.02
	MMRE	0.72	0.76	0.81	**0.72**	0.68

Figure 4.3 illustrates the sliding window performance for Project 1. MAE and MSE values fluctuate across the windows, with higher errors in the early and middle windows and lower values toward the later ones. The R^2 scores remain close to zero with only slight variation between windows. MMRE changes irregularly, without showing a consistent improvement trend. Overall, this project shows moderate reductions in error in later windows but does not demonstrate uniform improvement across all metrics.

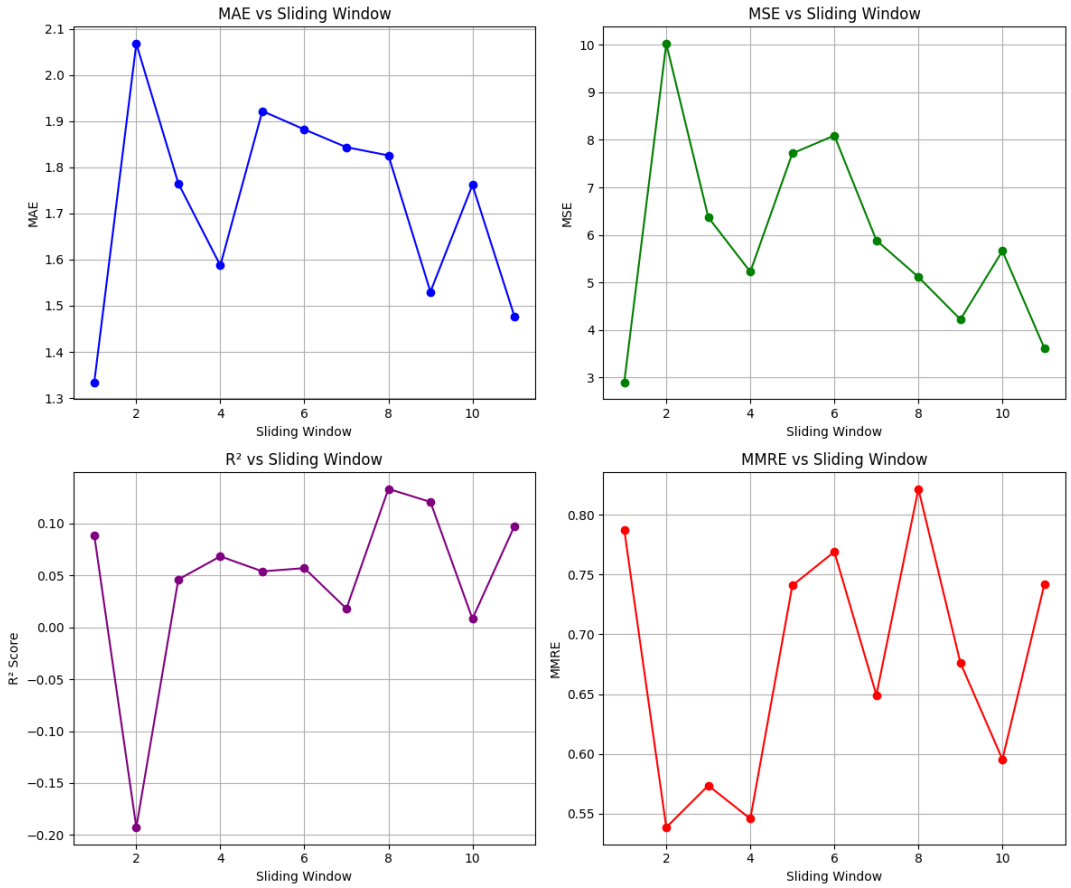


FIGURE 4.3: Performance metrics across sliding windows for Project 1.

Table 4.3 includes two additional projects (11 and 28) that were selected because they highlight behaviors not seen as clearly in the representative set. Project 11 is interesting, as it shows inconsistent patterns across metrics, with error measures and R^2 values peaking in different windows rather than aligning at a single point. Project 28, in contrast, demonstrates a steady improvement across several windows, particularly in terms of error reduction and relative error stability, making it a good example of a project where the sliding window approach reveals gradual gains over time. Together, these projects illustrate that while many projects follow the general trends observed earlier, not all projects share the same performance trajectory.

Table 4.3: Comparison of metrics across selected sliding windows (W1, W3, W5, W8, W11) for Projects 11 and 28.

Project ID	Metric	W1	W3	W5	W8	W11
11	MAE	2.33	2.46	**1.97**	2.02	2.31
	MSE	10.97	12.22	**5.81**	8.72	12.81
	R^2	0.020	-0.001	-0.263	**0.255**	-0.255
	MMRE	0.771	0.764	1.510	0.781	**0.549**
28	MAE	2.46	2.47	2.13	**2.18**	1.71
	MSE	12.05	12.30	9.65	10.18	**6.73**
	R^2	0.161	0.130	0.128	**0.213**	0.153
	MMRE	1.619	1.581	1.321	1.147	**1.031**

4.3 Concept drift detection and retraining

Table 4.4 summarizes the effect of retraining after detecting drift points in the representative projects. The table reports the number of drifts detected per project, the error before drift, the error after drift without retraining, and the error after drift with retraining. The final column shows the difference between the two conditions (No Retrain – Retrain), where positive values indicate that retraining reduced the error. Project 1 had no detected drifts and therefore shows no values. For the other projects, the number of detected drifts ranged from two to four. Positive improvements are observed in Project 3 and Project 13, where retraining lowered the error compared to not retraining, while the projects 12 and 36 show negative values, indicating higher error after retraining.

Table 4.4: Drift handling summary for representative projects. Positive improvement values indicate retraining reduced error.

Project ID	#Drifts	Before	No Retrain	Retrain	Improvment
1	0	–	–	–	–
3	2	0.97	1.05	0.74	**0.31**
12	2	0.98	0.65	0.86	**-0.21**
13	2	0.43	0.53	0.43	**0.10**
36	1	0.48	0.66	0.81	**-0.15**

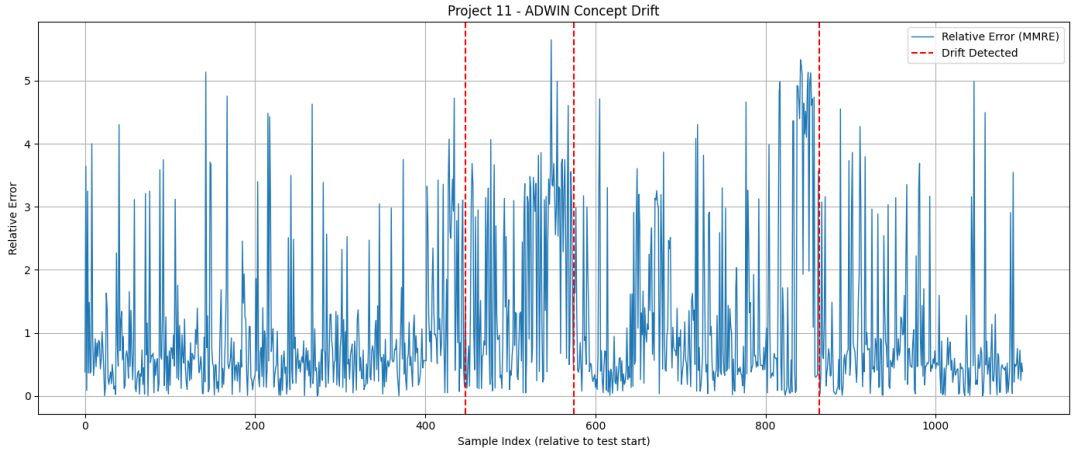
Table 4.5 presents drift handling results for two additional projects (11 and 28) not included in the representative set. Both projects demonstrate clear improvements when retraining is applied after drift detection. Project 11, with three detected drifts, shows a notable reduction in error after retraining, while Project 28, with twelve detected drifts, records the largest improvement, reflecting consistent benefit from retraining across multiple drift points. These cases are included as further examples where retraining leads to substantial gains compared to continuing without adaptation.

Table 4.5: Drift handling for two representative projects not used before. Positive improvement values indicate retraining reduced error.

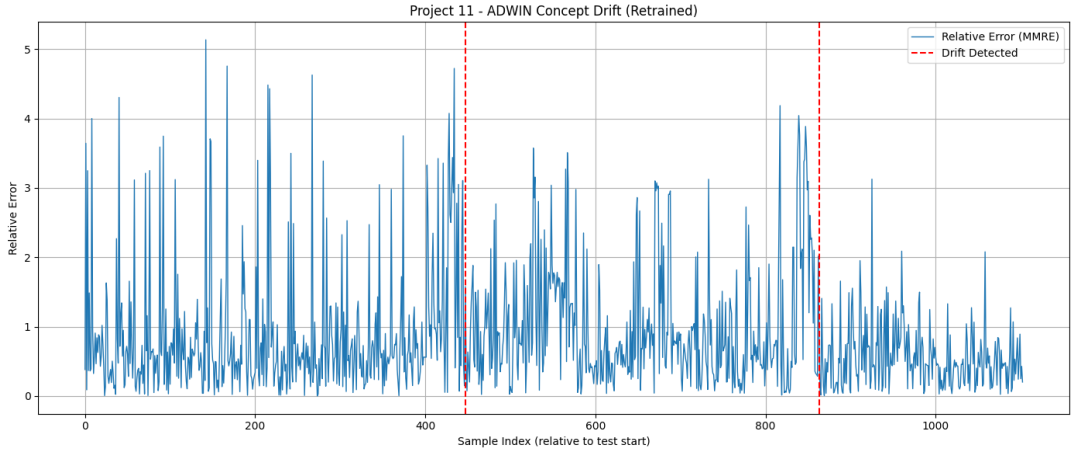
Project ID	#Drifts	Before	No Retrain	Retrain	Improvement
11	3	2.50	1.37	0.77	**0.60**
28	12	4.76	2.88	1.65	**1.23**

Figure 4.4 presents the results of drift detection for Project 11 before and after retraining. A clear difference can be observed between the two cases: without retraining, more drift points are detected and the relative error remains high across several regions, whereas after retraining fewer drifts are detected

and the overall error is considerably reduced. In both plots, the blue line represents the relative error (MMRE) over time, while the red dashed vertical lines mark the points where ADWIN signaled concept drift. The first plot (without retraining) shows frequent spikes in error and multiple drift detections, while the second plot (with retraining) displays a smoother error curve with lower overall values. Comparing the two, retraining not only reduces the magnitude of error but also stabilizes the model, resulting in fewer detected drifts and improved performance across the timeline.



(A) Concept drift detected without retraining.



(B) Concept drift with retraining applied.

FIGURE 4.4: Relative error for Project 11 using ADWIN drift detection, comparing (a) no retraining and (b) retraining. Red dashed lines indicate detected drift points.

Chapter 5

Discussion

In this chapter, the focus shifts from presenting results to analyzing and interpreting them in light of the research questions. The aim is to draw together the main observations from the experiments and to discuss what they reveal about the behavior of effort estimation models over time.

5.1 RQ1: Does concept drift occur in the context of effort estimation for IT projects?

5.1.1 Temporal folds

The temporal evaluation of effort estimation models across multiple datasets reveals consistent signs of concept drift, though the way it manifests differs between projects. In Project 1, training exclusively on the oldest projects produced weak predictive performance on the most recent cases, as shown by low R^2 values and relatively high MSE. As more recent data was incorporated, both MSE and R^2 improved, suggesting that newer projects better reflected the current estimation environment. Yet, MAE remained flat and MMRE rose after about 40% of the training history was included, indicating that outdated projects not only contributed little predictive value but in fact distorted relative accuracy. This pattern points to shifts in the relationship between project features and effort over time, with older practices or scales no longer matching contemporary conditions.

Project 3 displayed even stronger evidence of drift. As more historical data was added, both MAE and MSE worsened, and R^2 stayed negative until very recent projects dominated the training set. This shows that older projects actively harmed absolute predictive accuracy, a hallmark of substantial drift. Interestingly, MMRE moved in the opposite direction, declining steadily across the folds. This divergence suggests that while models trained on older projects struggled to predict absolute effort levels, they were still partially aligned with relative proportions, perhaps because early projects shared structural similarities with later ones even if their absolute scales differed. One plausible explanation is that Project 3 experienced significant shifts in development size or resource allocation practices, so relative trends remained consistent while absolute magnitudes changed. This duality underscores the importance of examining multiple metrics, since drift may appear differently depending on whether absolute or relative accuracy is emphasized.

Project 12 offers a contrasting but complementary perspective. With only the oldest projects in training, errors were high and R^2 was negative, but as newer projects were added, performance improved dramatically: MAE and MSE fell sharply, and R^2 became positive and stabilized. At the same time, MMRE increased from its lowest point and fluctuated across the folds, suggesting that while absolute accuracy benefited from contemporary data, relative errors remained unstable. This could reflect an evolving project portfolio in which the scale of tasks changed — for example, if later projects were

smaller, more modular, or differently structured, the relative distribution of effort would shift even as absolute predictability improved.

The results from Project 13 were more gradual but again consistent with drift. Initially, when trained only on the oldest projects, MAE and MSE were high and R^2 negative, but all three metrics steadily improved as more recent projects were included. MMRE, however, rose early and remained volatile through the middle folds before decreasing once the training set was dominated by recent data. This indicates that older projects obscured the underlying predictive relationships, and stability was only restored when the model relied more heavily on contemporary information. Here, drift seems to reflect a slow but steady evolution in project practices, where alignment with recent cases improved continuously as the temporal gap between training and testing narrowed.

Not all projects, however, showed such unambiguous signals. Project 36 provides a more cautious case, where the absolute error metrics suggested stability rather than strong drift. MAE and MSE both decreased steadily, and R^2 rose smoothly from near zero to positive values, implying that additional historical data was consistently beneficial. This could be interpreted as evidence against major drift, since the model appeared to generalize across the temporal folds. Yet, MMRE told a different story: relative error was lowest when only the earliest projects were included, worsened when mid-period projects were added, and only improved again when the training set became dominated by recent projects. One possible explanation is that older and recent projects in this dataset were more similar in absolute size than in others, so absolute accuracy benefited from longer histories, but subtle shifts in project composition or effort scaling created relative instability. This case highlights that drift does not always manifest strongly across all metrics and that its detection can be subtle, requiring attention to multiple indicators.

Beyond the representative projects discussed in detail, the remaining datasets provided a broader picture of how concept drift manifests with varying intensity. Some projects, such as the projects 4, 5, and 43, showed only subtle improvements in error measures, with drift signals confined mostly to small declines in MAE or MMRE, suggesting that these datasets were relatively stable across time. Others displayed much stronger and clearer drift patterns, as in the projects 8 and 29, where error metrics decreased monotonically and R^2 steadily improved, indicating that older data quickly became obsolete while newer data aligned closely with recent test cases. A different variant emerged in Project 11, where performance first improved but then deteriorated again in later folds, pointing to multi-phase drift where projects underwent several waves of change. Still other cases, such as the projects 24 and 34, revealed marked reductions in error metrics but consistently poor or negative R^2, suggesting that while temporal alignment reduced raw prediction errors, the explanatory power of the model remained limited — perhaps due to noisy or weak predictors. Project 25 added a more gradual picture, with modest yet consistent improvements across folds, while Project 28 highlighted the importance of metric choice: although MAE and MSE improved steadily, MMRE was distorted by project scale and produced abnormally high values. Taken together, these additional projects confirm that drift is not uniform: it can be strong or weak, monotonic or episodic, metric-dependent or error-driven. Yet across all cases, the common theme was that reliance on outdated data rarely improved performance, while incorporating more recent projects tended to stabilize or enhance predictive accuracy.

5.1.2 Sliding window

Building on the temporal folds evaluation, a sliding-window procedure was used to further investigate how models generalize across consecutive periods of project data. This approach captures short-horizon dynamics by training on a contiguous block of recent projects and testing on the immediately following block, allowing us to observe how predictive accuracy evolves locally over time. The results across projects again reveal strong evidence of temporal instability and concept drift, though with different

forms of manifestation.

For Project 1, performance varied considerably across windows. Early segments produced poor results, with high MSE and negative R^2, indicating strong mismatches between adjacent blocks. As the window advanced, however, performance improved, with MAE and MSE steadily declining and R^2 becoming consistently positive. Yet MMRE fluctuated sharply, swinging between unusually low and high values. This suggests that while absolute accuracy improved when relying on temporally close data, relative accuracy was more sensitive to changes in project scale or composition. One likely explanation is that Project 1 involved variations in team size or scope intensity over time — factors that may not alter the absolute prediction structure but shift the proportional distribution of effort, thereby destabilizing MMRE.

Project 3 showed even greater volatility, reinforcing its characterization as a dataset with strong drift. MAE and MSE spiked in certain early and middle windows, but dropped to very low values in others, such as around window 5. R^2 remained negative through much of the evaluation, only turning positive when specific windows aligned more closely. MMRE displayed similar instability, with peaks above 1.0 in early periods but stabilization closer to 0.5 in later windows. This inconsistency suggests that Project 3 underwent substantial shifts in practices or effort magnitudes, making some temporal segments predictive while rendering others obsolete. A possible explanation is that the dataset contains both large-scale projects with heavy resource use and smaller, more modular efforts. When windows happened to span periods dominated by one type, the following block matched poorly, causing severe error spikes. In other windows, structural similarities between consecutive projects allowed the model to generalize well, producing sharp error drops.

In Project 12, the first sliding window yielded extremely poor performance, with MAE and MSE at their highest and R^2 strongly negative. As the windows advanced, however, performance improved dramatically. Errors fell sharply, and R^2 turned positive and remained stable, indicating that more recent project blocks captured patterns that generalized better to their successors. MMRE, by contrast, fluctuated heavily, peaking around window 4 and dipping to its lowest value near window 6 before oscillating again. These fluctuations could be explained by shifts in project portfolio composition — for example, the introduction of new methodologies or technologies in mid-period projects that were later replaced or adjusted. Such structural changes would alter the relative accuracy of predictions even as overall absolute alignment improved.

Project 13 revealed a similar temporal dependence but with a clearer trajectory. In the earliest windows, performance was very poor, with high errors and strongly negative R^2. As the window advanced, however, both MAE and MSE declined sharply, stabilizing at low levels by the middle windows, while R^2 turned positive and remained consistently above zero. MMRE mirrored this pattern, falling from above 0.5 to a low of around 0.3, but rising again in the final windows. This trajectory implies multiple phases of drift: an early mismatch between old and new data, a period of stability in the middle, and renewed changes toward the end. One plausible explanation is that Project 13 reflects organizational or domain-level changes in effort estimation, such as shifting from large monolithic projects to shorter, more incremental releases, before moving again to new forms of work in later periods. These changes would naturally cause waves of alignment and misalignment in predictive accuracy.

Finally, Project 36 presented a more complex case. While MAE and MSE generally improved in the later windows and R^2 became positive, the early and middle periods displayed considerable instability, with performance fluctuating sharply from one block to the next. MMRE in particular was highly erratic, oscillating between relatively low and very high values, with no clear stabilizing trend. This suggests that Project 36 included both periods of continuity and abrupt shifts, so that some sliding windows aligned well with their test segments while others revealed pronounced mismatches. One possible explanation is that the dataset mixes heterogeneous project types — for example, maintenance tasks

alongside full development projects. When consecutive windows spanned projects of different natures, predictive accuracy deteriorated; when they spanned similar types, it improved. This helps explain why Project 36 appeared more ambiguous in the expanding-window analysis: absolute metrics smoothed over the variation, while the sliding-window view exposed localized bursts of instability.

Project 11 exhibited one of the most unstable sliding-window patterns, highlighting the episodic nature of concept drift. Both MAE and MSE fluctuated widely across the windows, with peaks of poor performance around windows 4–5 followed by dramatic improvements at window 6, where errors dropped to their lowest levels. R^2 remained negative in many windows but showed brief surges into positive territory when the training and test windows were better aligned, only to fall again when distributions shifted. MMRE followed the same instability, with a spike above 1.4 in window 5 but more moderate values near 0.6 toward the end. This volatility suggests that Project 11 underwent abrupt shifts in project scale or practices that disrupted predictive performance. One plausible explanation is that the dataset spans distinct phases of organizational change, such as a transition to new development methodologies or shifts in project size norms, which would create sharp mismatches between consecutive windows. The sliding-window analysis therefore shows that drift in Project 11 is less gradual and more episodic, with alternating periods of alignment and severe misalignment across time.

Project 28 produced results that differ from the other datasets and highlight the complexities of detecting drift. MAE and MSE were relatively stable in the first half of the windows and then decreased in later ones, suggesting gradual improvement as training data moved closer in time to the test blocks. Unlike most projects, R^2 remained positive throughout, indicating that predictive models consistently captured at least some variance in the target variable across all windows. The most striking result, however, was the MMRE behavior: relative error remained very low and flat across most windows, only to spike dramatically in window 10. This outlier suggests that the corresponding test block contained unusually small actual effort values, which can inflate relative errors even when absolute predictions are close. Project 28 therefore represents a case where drift is less about global distributional changes and more about sensitivity to particular data slices. It illustrates that metrics like MMRE can exaggerate drift effects when effort magnitudes vary widely, underscoring the importance of interpreting results across multiple error measures rather than relying on a single indicator.

Together, the projects 11 and 28 add nuance to the sliding-window experiment by showing alternative ways in which drift can appear. Project 11 highlighted the episodic nature of drift, with sharp spikes of error followed by rapid recoveries, likely reflecting organizational or methodological shifts that disrupted estimation patterns in some periods but not others. Project 28, by contrast, presented a case of metric sensitivity, where absolute accuracy was stable and even improved, while relative error suddenly spiked due to scale effects in one test block. These cases show that concept drift in effort estimation does not always manifest as a smooth deterioration in predictive accuracy. Instead, it can appear as bursts of instability linked to organizational changes (Project 11) or as artifacts of shifting project scales that distort certain error measures (Project 28). Including such cases therefore enriches the overall analysis, making clear that drift is multifaceted and must be interpreted in light of both project context and the chosen evaluation metrics.

In the sliding-window analysis, the remaining projects showed varied patterns of drift. Some, such as the projects 4, 5, and 43, exhibited relatively minor changes in MAE and MSE across consecutive windows, suggesting stable dynamics with only modest fluctuations in R^2 or MMRE. By contrast, Project 26 experienced sharp breakdowns in certain windows, with error spikes and strongly negative R^2 values that pointed to abrupt distributional changes. Project 29 showed the opposite, with a steady decline in error metrics and lower MMRE in later windows, indicating gradual improvement as training data became more recent. Projects 24 and 34 displayed marked reductions in MAE and MSE but weak or unstable R^2, reflecting improved absolute accuracy without stronger explanatory power. Project 25 lay between these

extremes, with modest improvements tempered by instability in later windows. Overall, these results highlight that under sliding windows, drift often appears as localized instability - sometimes gradual, sometimes abrupt - but in most cases, models trained on more recent data generalized better to the following projects.

Taken together, the sliding-window experiment reinforces the evidence of concept drift while highlighting its local and episodic nature. Unlike the expanding-window design, which emphasized how older data gradually loses relevance, the sliding-window results reveal how predictive performance can vary sharply across consecutive periods, with some windows producing strong generalization and others reflecting sudden mismatches. Across projects, absolute error metrics tended to improve as the training blocks approached the test segments, while relative errors were more volatile, capturing shifts in project scale and distribution. These findings suggest that drift in IT project effort estimation is not only a long-term phenomenon but also a short-horizon challenge, shaped by shifts in practices, project portfolios, or methodologies. Detecting it therefore requires attention to both absolute and relative measures across localized temporal windows.

5.1.3 Concept drift detection

Applying ADWIN without retraining revealed a wide range of drift signals across the projects, though the impact on error was inconsistent. In some cases, such as the projects 3 and 34, flagged drifts coincided with clear performance deterioration, with mean error increasing after detection, which is consistent with the expectation that distributional changes undermine predictive accuracy when the model is not updated. In the other projects, such as 11 and 25, ADWIN marked points where error actually declined afterwards, suggesting that the detector was also sensitive to transitions where the following data segment was easier for the static model to predict. Project 4 and Project 13 showed mixed results, with some detections linked to worsening accuracy and others to improvements, reflecting the uneven impact of drift within the same dataset. Project 28 was particularly striking, as ADWIN triggered a large number of detections, some associated with dramatic error reductions (for example from over 1800 to close to 1) while others coincided with increases, showing how the detector responded strongly to this dataset's volatility even when not all flagged changes reflected meaningful long-term drift. The other projects, such as 12, 24, 36, and 43, showed only very small error shifts around detected points, suggesting that ADWIN was capturing subtle distributional changes that had limited practical impact on predictive error.

Dataset size also had a strong influence on ADWIN's sensitivity. Projects with very few rows, such as 5 with 464 issues, 8 with 696, and 29 with 634, did not produce any detections. With so little data, ADWIN had fewer opportunities to identify statistically significant shifts, even though earlier experiments suggested some drift was present. By contrast, large datasets such as Project 28 with 17,629 rows produced numerous detections, reflecting both genuine distributional shifts and the detector's greater statistical power when more data is available. Mid-sized datasets fell between these extremes, sometimes producing a small number of detections such as Project 11 with 3, and sometimes only one, such as Project 36 or Project 43. Project 1 was a notable exception: despite having over 3,300 rows and showing drift in earlier experiments, ADWIN did not flag any changes. This suggests that drift in Project 1 was gradual rather than abrupt, and therefore fell below the statistical threshold that ADWIN uses to signal a change.

Taken together, these results confirm that ADWIN can identify distributional change points in many datasets, but without retraining its value for improving predictive accuracy remains limited. Some detections corresponded to harmful drift, others to benign or even helpful transitions, and in some cases no detections occurred at all, either due to dataset size constraints or because the drift was too gradual

to be picked up. This reinforces the view that detection alone is insufficient for effort estimation and must be paired with adaptive retraining to be effective.

5.1.4 Does concept drift appear in effort estimation?

The results from all three experiments provide consistent evidence that concept drift does occur in the context of effort estimation, though its manifestation varies. Temporal folds showed that in most projects, predictive accuracy improved when more recent data was included, indicating that older data often became obsolete over time. Sliding windows reinforced this, but highlighted that drift is not always gradual: in many projects performance fluctuated sharply between consecutive windows, with phases of alignment, collapse, and recovery, suggesting that drift can be episodic and unstable rather than strictly monotonic. ADWIN confirmed that distributional changes were present in many datasets, particularly in larger and more volatile projects, but it also revealed limitations: some projects with gradual drift or small datasets were not flagged, and in others detections did not consistently correspond to worsening error. Overall, the three approaches together show that concept drift is a recurring phenomenon in IT project effort estimation. Its form ranges from gradual degradation to abrupt shifts or episodic instability, but the common thread across experiments is that models trained on recent data consistently outperformed those relying on older history, underscoring the temporal sensitivity of effort estimation.

5.2 RQ2: If concept drift occurs, can it be effectively counteracted using existing strategies such as drift detection and model retraining?

The addition of retraining at drift points revealed that adaptation can, in many cases, restore predictive accuracy, but the effectiveness of this strategy varied strongly between projects. Several datasets demonstrated clear benefits. In project 3, for example, retraining after the first detected drift reduced mean error from 1.10 to 0.71, and in Project 11 error fell even more sharply, with reductions from 2.03 to 0.76 and from 1.92 to 0.42 across different drift points. These cases illustrate the potential for retraining to re-align the model with new data distributions. Similarly, in the projects 24 and 25, retraining cut error by more than half, confirming that targeted updates can significantly improve accuracy in situations where drift undermines static models.

The results from larger datasets, particularly Project 28, further underline the promise of this approach. Without retraining, ADWIN detections often coincided with erratic error changes, sometimes even dramatic collapses. With retraining, however, several of these detections turned into strong improvements: error dropped from 6.45 to 0.56 at one drift point, from 6.87 to 2.44 at another, and most notably from an extreme outlier value of 874 to just over 1 after retraining. These outcomes suggest that large, volatile datasets benefit substantially from adaptive updates, as they are more likely to contain severe distributional changes that render the old model obsolete.

At the same time, the experiments also revealed limitations. Not all retraining attempts improved accuracy. In Project 3 at index 1055 and in Project 4 at index 2239, retraining actually increased error. Project 12 showed mixed results as well, with one drift point worsening error (0.63 to 0.97) and another improving it (1.32 to 0.75). Project 28, despite its overall gains, also included several retraining points where error increased rather than decreased. These cases highlight that retraining is not automatically beneficial; if the retraining window is too small, noisy, or unrepresentative, it can lead to overfitting or destabilize the model.

Other projects provided additional nuance. Project 13 saw error increase after retraining at the first detection but improve at the second, reflecting the uneven impact of retraining depending on the specific data shift. Project 34 and Project 36 showed similar inconsistencies, with retraining sometimes

worsening performance despite evidence of drift. Project 43, by contrast, exhibited only a slight change after retraining, reflecting the limited effect of drift in more stable datasets.

5.3 Summary of findings, their implications and limitations

Taken together, these results suggest that drift detection and retraining can be effective strategies for counteracting concept drift in effort estimation, but their success is conditional. Retraining works best in projects with large amounts of data and clear distributional shifts, where the benefits of updating outweigh the risks of noise. In smaller or more stable datasets, retraining may add little value or even degrade performance temporarily. Compared to the earlier experiments, retraining clearly improved upon detection alone by turning many drift points into opportunities for recovery, yet it also introduced new variability. The key implication is that while existing strategies can mitigate drift, their deployment must be sensitive to dataset size, data quality, and the nature of the drift. Effective countermeasures therefore require not only drift detection and retraining, but also careful design of update policies that balance responsiveness with stability.

The findings of this thesis carry important theoretical and practical implications. Theoretically, they show that effort estimation models should be seen as evolving systems rather than static predictors, supporting earlier observations of temporal instability while challenging the assumption that more historical data always improves accuracy. By explicitly framing these changes as concept drift, the thesis connects software engineering with established theories in machine learning. Practically, the results caution against heavy reliance on outdated datasets, since older information can reduce predictive performance, and they demonstrate that adaptive strategies such as drift detection with retraining can often restore accuracy, particularly in large and volatile projects. However, their effectiveness was inconsistent, highlighting the need for careful update policies in practice. The work also has limitations: some datasets were small, reducing statistical power, and the reliance on open-source projects constrains generalizability to industrial contexts. In addition, it is worth noting that this study considers concept drift only in terms of its impact on estimation accuracy. However, if drift occurs, it must have underlying causes, such as changes in the type of project work or in the way requirements are defined. This perspective was not examined here, but exploring it could help explain why some adaptive strategies succeed in certain projects while failing in others. Understanding the origins of drift therefore represents both a limitation of this research and a promising avenue for future studies. Future research should also test alternative drift detection methods, explore richer project features, and evaluate adaptation strategies on industrial datasets, with longitudinal studies assessing retraining policies across full project lifecycles. These directions would help establish effort estimation as a continuous, adaptive process rather than a one-off modeling exercise.

This thesis demonstrates that concept drift is a recurring and significant challenge in software effort estimation. The experiments showed that prediction accuracy declines when models rely on outdated data, that drift often appears in abrupt or episodic forms rather than gradual ones, and that drift detection with retraining can in many cases restore performance, especially in large and dynamic datasets. At the same time, the results highlight that adaptation is not universally reliable and depends on factors such as dataset size, stability, and the representativeness of the retraining data. The overall significance of these findings is that effort estimation must be treated as a time-sensitive and adaptive process rather than a static task.

5.4 Threads to validity

As with any empirical study, this work is subject to several threats to validity, which can be grouped into three main categories:

- Internal validity – the design choices in data preparation and experimentation may have influenced the outcomes. For example, the selection of error metrics (MAE, MSE, R^2, MMRE) emphasizes certain aspects of model performance and may overlook others. Similarly, the choice of Random Forest as the primary learning algorithm, along with fixed parameter grids, could bias the results toward particular behaviors. While these choices were made to ensure consistency across experiments, they may limit the generality of specific performance observations.

- External validity – the generalizability of the findings is constrained by the datasets used. All datasets were drawn from open-source projects (e.g., TAWOS), which may not fully represent the dynamics of industrial or proprietary environments where processes, team structures, and governance differ significantly. Smaller datasets (such as the projects 5, 8, and 29) further limit the robustness of conclusions, as ADWIN and retraining strategies have less statistical power in such cases. Future validation on industrial datasets would help establish broader applicability.

- Conclusion validity – while the experiments provided consistent trends across many projects, variability in outcomes—particularly in ADWIN with retraining—indicates that conclusions must be drawn with caution. The statistical tests embedded in ADWIN increase confidence that detected changes are meaningful, but they do not fully rule out spurious detections or missed gradual drift. Similarly, observed improvements or deteriorations after retraining depend on specific segment boundaries, which may not generalize across all contexts.

In summary, although these threats may limit the scope of generalization, they do not undermine the central finding: that concept drift is a recurring factor in software effort estimation, and that drift-aware strategies, while promising, are not universally reliable.

Chapter 6

Conclusion

This thesis set out to investigate whether concept drift occurs in the context of software effort estimation and, if so, whether it can be effectively addressed through existing adaptation strategies. Three complementary experiments were conducted: temporal folds, sliding windows, and adaptive windowing (ADWIN), both with and without retraining. Together, these provided a comprehensive view of how predictive models behave when confronted with evolving project data.

The results demonstrated that concept drift is not an occasional anomaly but a recurring and significant factor in effort estimation. Across projects, predictive accuracy consistently declined when models relied on outdated data, while more recent information improved performance. Drift manifested in different forms — sometimes gradual, sometimes abrupt, and often episodic — underscoring the dynamic nature of software projects. Importantly, the experiments with ADWIN showed that drift can be statistically detected, and retraining at drift points frequently restored performance, particularly in large and volatile datasets. At the same time, adaptation was not universally reliable: in some cases, retraining led to increased error, highlighting the risks of small or unrepresentative training windows.

These findings contribute to theory by reframing temporal instability in effort estimation as a problem of concept drift, thus connecting software engineering with the broader machine learning literature on evolving data. Methodologically, the study demonstrates how drift-aware evaluation strategies can be applied in effort estimation research, moving beyond static train-test splits that mask temporal effects. Practically, the results warn practitioners against over-reliance on long historical datasets and suggest that adaptive retraining policies may help sustain accuracy in dynamic environments, although careful design and validation are required.

Naturally, the work has its limitations. The reliance on open-source datasets constrains generalizability to industrial contexts, and smaller datasets limited the statistical power of drift detection. The choice of metrics and algorithms may also have shaped the results, and further replication with different learners and error measures would strengthen the conclusions.

Future research should build on these foundations by experimenting with alternative drift detection and adaptation strategies, incorporating richer project features, and testing approaches in industrial datasets. Studying project lifecycles over time and testing hybrid strategies that balance flexibility with stability could yield valuable insights.

In summary, this thesis has shown that concept drift is a central challenge in software effort estimation and that while adaptation can mitigate its effects, it is not a complete solution. The main takeaway is that effort estimation should no longer be treated as a static modeling task but as a continuous, adaptive process that evolves with the projects it seeks to predict.

Bibliography

[1] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, and Bert Steece. *Software Cost Estimation with COCOMO II*. 2000.

[2] Kjetil Moløkken-Østvold and Magne Jørgensen. A review of surveys on software effort estimation. In *International Symposium on Empirical Software Engineering (ISESE)*, 2003.

[3] Muhammad Mahmood, Bashir Shahzad, Muhammad Idrees, Awais Mahmood, Iqra Ashraf, Muhammad U. Sarwar, and Muhammad Shahzad. Systematic literature review on software effort estimation using machine learning algorithms. *Journal of Software: Evolution and Process*, 2021.

[4] Shabana Rathore, Deepak Kumar, and Riyaz Ahamed Khan. A systematic literature review of machine learning techniques for software effort estimation. *Journal of Systems and Software*, 2022.

[5] Sumeet Kaur Sehra, Yadwinder Singh Brar, Navdeep Kaur, and Sukhjit Singh Sehra. Research patterns and trends in software effort estimation. *Information and Software Technology*, 2017.

[6] Vali Tawosi, Afnan Al-Subaihin, and Federica Sarro. Investigating the effectiveness of clustering for story point estimation. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022.

[7] Vali Tawosi, Rebecca Moussa, and Federica Sarro. On the relationship between story points and development effort in agile open-source software. In *ESEM*, 2022.

[8] João Gama, Indrė Žliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 2014.

[9] Indrė Žliobaite. Overview of concept drift applications. *ACM Computing Surveys*, 2016.

[10] Geoffrey I. Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. Understanding concept drift. *ACM Computing Surveys*, 49(4), 2016.

[11] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing (adwin). In *SIAM International Conference on Data Mining (SDM)*, 2007.

[12] Fabian Hinder, Valerie Vaquet, Johannes Brinkrolf, and Barbara Hammer. Model-based explanations of concept drift. *Neurocomputing*, 555, 2023.

[13] Chris Lokan and Emilia Mendes. Applying moving windows to software effort estimation. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2009.

[14] Martin Shepperd and Chris Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12):736–743, 1997.

[15] Zheng Li, Xin Jing, and Li Zhang. Towards a theoretical framework for software cost estimation. *Information and Software Technology*, 87:1–15, 2017.

[16] Chris F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, 1987.

[17] Magne Jørgensen and Martin Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, 2007.

[18] Adam Trendowicz and Ross Jeffery. *Software Project Estimation*. Springer, 2014.

[19] Mike Cohn. *Agile Estimating and Planning*. Pearson Education, 2005.

[20] Leandro L. Minku and Xin Yao. Can cross-company data improve performance in software effort estimation? In *International Conference on Predictive Models in Software Engineering (PROMISE)*, 2012.

[21] Leandro L. Minku and Xin Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, 2013.

[22] Vali Tawosi, Afnan Al-Subaihin, Rebecca Moussa, and Federica Sarro. A versatile dataset of agile open source software projects. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 707–711, 2022. See also the project repository at `https://github.com/SOLAR-group/TAWOS`.

[23] ADWIN (adaptive windowing) - drift detection - river documentation. `https://riverml.xyz/dev/api/drift/ADWIN/`. Accessed: 2025-07-22.

Attachments

All experimental code, results, and plots used in this thesis are openly available in the accompanying GitHub repository at github.com/julia-mula/concept-drift.

Poznan University of Technology
Faculty of Computing and Telecommunication
Institute of Computing Science

Typeset using LaTeX.