Monitoring Effort Overruns in Software Maintenance

Mariela Pluutus (s3816281)

Supervisors:
Prof. dr. ir. J.M.W. Visser – Leiden University
Dr. C.J. Stettina MSc – Leiden University
E. de Kruif – Netcompany

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl
June 9, 2026

Acknowledgment

I would like to thank my supervisors, Prof. dr. ir. J.M.W. Visser and Dr. C.J. Stettina, for their guidance, and practical feedback throughout this thesis project. Their input helped me refine and sharpen the direction of the study and improve the quality of the research. I would also like to thank A. Imamoglu, E. de Kruif, and Netcompany Netherlands B.V. for the opportunity to conduct this thesis within the APS department. Being included in a real agile maintenance environment gave me firsthand experience of how software teams work in practice, and I am grateful for the access to the case-study data and the openness to let me explore their processes. I would also like to thank the practitioners who took the time for interviews to share their experience with effort estimation, re-estimation, and client communication, as their insights were valuable for grounding the quantitative findings.

Abstract

Context: In client-facing software maintenance projects, client requested changes receive an effort estimate before implementation. Once approved, this estimate becomes the authorized budget for completing the work. However, estimating maintenance tasks is difficult by nature due to hidden complexities, evolving requirements, and unknown dependencies in existing systems. Consequently, tasks may exceed their approved estimates and require re-estimation, planning adjustments, or renewed client approval before additional effort can be authorized. Although project management systems routinely record effort estimates, logged time, and estimate updates throughout development, this information is not always used to actively monitor ongoing tasks, and their overrun risk. As a result, overruns often become visible only once the approved estimate has already been exhausted, leaving limited opportunity for proactive intervention.

Objective: This study investigates whether task-level effort estimation and effort-consumption data contain sufficient information to identify ongoing tasks at risk of exceeding their approved effort budget before the estimate is exhausted. Using a five-year software maintenance project as a case study, historical task data is first analyzed to quantify estimation accuracy, the frequency and magnitude of estimate overruns, and whether re-estimation occurs early enough to support proactive planning and client communication. The study then evaluates whether this data can support an automated early-warning approach using either a Logistic Regression classifier or a simple effort-consumption threshold to raise awareness to and identify at-risk tasks before the approved estimate is exceeded.

Method: We conduct an empirical quantitative case study using anonymized data from a single software maintenance project. The dataset contains approximately 7500 tasks with reconstructable version histories. These histories are analyzed to examine estimation practices and to develop and evaluate an exploratory Logistic Regression classifier for overrun-risk detection. Its performance is compared with a simple effort-consumption threshold rule, and additional semi-structured practitioner interviews are conducted to validate and contextualize the findings.

Results and Conclusion: Underestimation is common in the studied maintenance project: 53.4% of completed Change cases exceed their original estimates, and 14.4% of all recorded effort is logged beyond the active estimate. Re-estimation is also common, however, 46.8% of estimate updates occur only after the active estimate has already been exceeded. The Logistic Regression model demonstrates that routinely collected project management data contains meaningful predictive information for identifying overrun risk, although most of this signal is captured by the effort-consumption ratio. A simple 75% consumption threshold performs nearly as well as the trained model while requiring only the approved estimate and cumulative logged effort. Since formal estimate updates often depend on internal review and client approval, the main challenge is not only estimation inaccuracy but also limited visibility into effort consumption and the absence of structured review points before the budget is exhausted. The study therefore recommends introducing simple 75% and 90% consumption-based review triggers, organized into a tiered risk categorization of ongoing tasks by consumption ratio, supported by visual and automated notifications and process improvements for effort monitoring and documentation, to reassess remaining work and initiate client communication before the approved estimate is consumed.

Contents

1 Introduction

Effort is one of the most important metrics to measure in software projects [1]. Effort estimation refers to predicting the amount of effort required to complete a project, task, or work item, and is used to directly support project planning, budgeting, team allocation, and task prioritization [23, 8]. In client-facing maintenance contexts, requested changes are often estimated before implementation begins. The resulting estimate defines the expected scope and effort, supports delivery expectations, and forms the basis for the budget communicated to and approved by the client. When actual effort exceeds the approved estimate, the consequences depend on the contractual and organizational context. In fixed-price arrangements, additional effort may reduce project margins and pressure teams to cut scope or accept lower quality solutions in order to still deliver within the agreed budget. In time-and-materials agreements, the client pays for the additional required effort, where late recognition of overruns can lead to unexpected budget increases and reduced trust for the client.

However, producing accurate estimates remains a challenge, as software work is often affected by underlying uncertainty from incomplete information, hidden complexity, changing requirements, and a lack of team experience [11, 18]. This challenge is especially relevant in software maintenance projects, where teams work on changes, system additions, bug fixes, or service requests for systems that have already been deployed. Maintenance tasks often require developers to understand existing code, identify technical dependencies, and handle unknown system behavior, especially when dealing with systems that may not have originally been developed by themselves.

As a result, the required effort may become clearer only as development progresses, causing the final cumulative effort recorded at completion to differ substantially from the initially predicted and anticipated estimates, making task-level estimation in maintenance work highly uncertain and prone to error. Previous research shows that effort estimates are frequently underestimated in practice, leading to effort overruns [24, 23]. In such cases, the required effort often becomes clearer only after development or implementation has started, meaning that current estimates may become less realistic as work progresses. If potential overruns are recognized too late, teams have less opportunity to react and re-estimate the task, adjust planning, and communicate with the client to request additional budget.

Many software organizations already collect the historical effort-related data needed to monitor risk during task execution in their administrative systems, such as effort estimates, recorded time spent, status changes, and re-estimation updates. In principle, these records make it possible to identify tasks at risk of overruns through tracking how much of the active estimate has already been consumed and how estimates evolve throughout the task life cycle. However, this monitoring potential is rarely being explored and utilized. Existing research focuses on effort prediction through algorithmic models [21, 15], machine learning [22], and hybrid approaches [1]. Additional recent work has explored early risk detection by comparing predicted task duration with planned effort [3]. However, it remains unclear from research whether actively collected effort-consumption data could warn teams when ongoing tasks are at risk of exceeding their current estimate.

Our research therefore examines how effort estimation and re-estimation are currently practiced in client-facing software maintenance projects, and whether routinely collected task-level data contains sufficient signal to support earlier identification of overrun risk. The study is conducted as a quantitative case study at Netcompany Netherlands B.V., using anonymized data from a five-year maintenance project with detailed version-level task histories, where all team member identifiers,

task titles, and case IDs are anonymized. First, we analyze how accurately initial estimates reflect final recorded effort and how frequently overruns occur during tasks. We then examine how often re-estimation occurs and at what point it takes place in relation to estimate consumption, in order to assess whether current practices are timely enough to support proactive planning and client communication. Building on this, we explore whether a model-based classifier or a simple consumption-ratio threshold rule can raise increased awareness of at-risk tasks before the approved estimate is exhausted, providing teams with measurable lead time to review remaining work, revise effort estimates, allocate team members, and initiate client communication before the budget is exceeded.

1.1 Research Question

This leads us to the main research question of this paper:

How can effort-consumption data be used to support earlier identification of ongoing software maintenance tasks at risk of exceeding their approved effort estimate?

To answer this main research question, we address the following sub-questions:

- **RQ1:** How accurate are early task-level effort estimates compared to the actual effort recorded, and how does estimation accuracy vary?
- **RQ2:** How often do active-estimate overruns occur during the life cycle of tasks, and how large are these overruns?
- **RQ3:** How often are effort estimates formally updated in the system, and at what level of effort consumption do these updates occur?
- **RQ4:** To what extent can historical effort-consumption data support classification of ongoing cases at risk of exceeding their active effort estimate?

1.2 Thesis overview

Section 2 provides the necessary background on effort estimation, overruns, re-estimation, and historical effort data. Section 3 reviews existing related work on effort estimation models and early risk detection. Section 4 describes the case-study environment and data source. Section 5 explains the methodology, including preprocessing, operational definitions, analyses, and model development. Section 6 presents and discusses the results, including threats to validity, and Section 7 concludes the study and suggests future work directions.

2 Background

This section provides the theoretical background for understanding effort estimation and its monitoring, active-estimate overruns, and re-estimation in Agile software maintenance projects. Section 2.1 introduces software effort estimation and explains its role in planning, budgeting, and task prioritization. Section 2.2 discusses effort overruns and the technical, human, and organizational factors that contribute to estimation error. Section 2.3 explains re-estimation as an operational decision and highlights why its timing matters for planning and client communication.

2.1 Effort Estimation in Software Development

Software Effort Estimation (SEE) is a key process in Agile Software Development (ASD), in which teams predict the effort required to complete a project, task, or work item. Software estimation can target various aspects of a project, including size, effort, schedule, and cost [8]. It is necessary as it directly helps lay the foundation for making successful investments for both software companies and clients [22], and guide the planning of the project, decision-making processes to successfully deliver high-quality products [15, 8]. For clients, estimates provide insight into the expected costs and delivery times for a project, allowing them to evaluate proposals and compare suppliers. Whereas for software companies, estimates help determine whether the requested work by the client is feasible, how many developers should be allocated, and how the work should be planned and divided into iterations to deliver the software product successfully.

In practice, projects are often first estimated in terms of size, meaning how large the expected work is based on the known requirements and client scope. This size estimate is then translated into an effort estimate, which can be documented differently depending on the estimation context and practice. More specifically, in agile settings, effort is often documented using relative measures such as story points, which Usman et al. found to be the most frequently used size metric among Agile practitioners [23]. Additionally, effort is often documented in time-based units, such as hours, days, or person-months [24], which can be connected to staffing allocations and developers' hourly rates, to calculate expected project costs.

The purpose and reliability of an estimate depends on when it is created. At the project level, estimates can be made at different stages of the project life cycle. Early rough estimates are used to support client negotiations, planning and risk measurement. However, it should be considered that such *pre-planning estimates* are usually created based on a broad understanding of the client request and the expected scope of work, serving purely as an initial indication of expected work rather than a precise prediction [14]. In some cases, these estimates are additionally influenced by a price-to-win strategy, where the estimate is influenced by the considered price that is most likely to be accepted by the client and competitive enough to win the project.

Once an agreement has been made with the client, requirements are made clearer, and project estimates are refined. At this stage, *detailed planning estimates* are created. These estimates can be made at the project, release, sprint, or task level, and are based on more knowledge than early pre-planning estimates. They are intended to support more informed planning decisions. However, even detailed estimates include uncertainty because software work can still be affected by emerging complexities throughout development.

Agile Software Development (ASD) follows an iterative approach in which small, cross-functional teams divide work into short development cycles, such as sprints or iterations [9]. Instead of defining

all requirements and planning decisions upfront, Agile teams continuously prioritise and refine requirements, plans, and estimates as more information becomes available throughout development. As a result, estimation is not only performed at a higher level at the beginning of a project, but also throughout the project during sprint and release planning, where individual work items are selected based on priority, expected effort, and available team capacity [23].

In ASD and maintenance projects, detailed planning estimates are especially relevant at the task level. Agile teams commonly divide work into smaller iterations or sprints, where individual work items are estimated and selected based on priority, expected effort, and available team capacity. In maintenance contexts, these work items often include change requests, bug fixes, service requests, or system additions. Estimating such tasks is difficult because the required effort depends not only on the requested change itself, but also on the existing code, technical dependencies and complexities, affected components, and required testing [21]. Therefore, within this study we closely focus on task-level estimation within a software maintenance context.

2.2 Effort Overruns: Human, Technical, and Organizational Causes

Estimation accuracy refers to how closely a predicted effort estimate for a given project task matches the actual effort required to successfully complete the work. In empirical studies, estimation accuracy is commonly assessed through estimation error, which refers to the difference between the initially predicted effort and the actual recorded effort after completion [18]. When the actual effort is higher than the estimated effort, the work has been underestimated, while when the actual effort is lower than the estimated effort, the work has been overestimated.

Both types of estimation error can negatively affect an organization. Underestimation may lead to the acceptance or planning of work that cannot realistically be completed within the agreed time or budget, which can result in duration overruns, cost overruns, reduced quality, and damage to the organization's credibility and reputation. Overestimation can also be harmful, as it may cause organizations or clients to reject work that would otherwise be valuable and manageable in practice. Prior research suggests that underestimation is a commonly recurring issue in software effort estimation. For example, Usman et al. [23] and Fernández-Diego et al. [8] report that Agile teams often experience underestimation due to overoptimism, team inexperience, and a tendency to only consider best-case scenarios.

Effort overruns can therefore be understood as the result of a combination of technical, human, and organizational factors. From a technical perspective, software maintenance work is often difficult to estimate because the required effort depends on the existing system, hidden dependencies, testing, and unexpected implementation complexity. These factors may only become visible once development has already started. As a result, the initial estimate may be based on incomplete knowledge of the actual work that will be required. At the same time, effort estimates are usually created through human judgment, which means they can be influenced by cognitive biases. Cognitive biases are deviations from optimal reasoning that can affect software engineering activities, including effort estimation [17]. A common bias is optimism or overconfidence. Software professionals often underestimate the effort required because they are overly confident in their skills and accuracy of their own estimates [13]. This is especially relevant in maintenance projects, where hidden complexity often becomes clear only after developers inspect the existing system and understand how a requested change affects the wider system.

Another important psychological factor is anchoring. Anchoring occurs when an initial value

or reference point influences later judgments, even when that value is not fully reliable. In software effort estimation, an early estimate, client expectation, budget indication, or previous comparable task can become an anchor for later estimation decisions. Løhre and Jørgensen show that numerical anchors can strongly affect software development effort estimates, even when the anchoring information is irrelevant or comes from a low-credibility source [16]. More general anchoring research also shows that initial numerical values can influence later judgments and that the strength of anchoring depends on how information is processed [20, 25]. This suggests that early estimates should be interpreted carefully, because they may continue to influence later decisions even when more information becomes available.

The combination of technical uncertainty, optimism, and anchoring can make effort overruns more likely. For example, a low initial estimate may become a reference point for planning and client expectations, while the team may also assume that the task can be completed under favorable conditions. If hidden complexity is discovered during implementation, the task may consume effort faster than expected. This means that the estimate is not only a numerical prediction, but also part of the organizational context for planning, budgeting, and client communication. Effort overruns are therefore operational problems. When a task consumes more effort than expected, the organization has to act accordingly and reallocate team capacity, adjust planning, revise the estimate, or request additional client approval. If the overrun is detected late, there is less opportunity to respond proactively. This makes it valuable to monitor effort consumption during the task life cycle.

2.3 Re-estimation as an Operational Decision

Effort estimates are uncertain by nature because they are predictions about future work made with often incomplete information. In software projects, the actual final cumulative effort required for a task may differ substantially from the initially estimated effort in the beginning due to unclear requirements, hidden technical complexity, changing scope, dependencies, team experience, and uncertainty about the existing system [22]. This is especially relevant in Agile settings, where requirements are constantly being refined throughout development when new information becomes available.

The cone of uncertainty, shown in Figure 1, illustrates the relationship between available information and estimation accuracy. Early estimates act like a rough guide. They are the least reliable as they are made when there exists limited information on the scope and requirements. As development progresses, more information becomes available, and estimates start to diverge as they become more realistic and reliable. From this perspective, estimation accuracy can be understood as largely knowledge-dependent: the more complete and reliable the information is, the more accurate are the effort estimations.

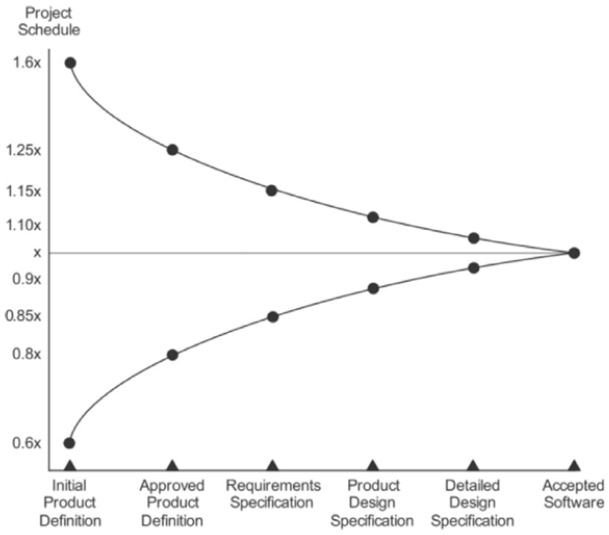


Figure 1: The cone of uncertainty [7]

When new information becomes available during task execution, the task may be re-estimated. Re-estimation refers to the process of revising an earlier effort estimate in order to reflect a more realistic expectation of the required work. In practice, re-estimation may occur when requirements are clarified, hidden dependencies are discovered, implementation is more complex than initially expected, or the scope of the work changes. Re-estimations can also decrease the original estimate, for example when the task turns out to be less complex than expected, the scope is reduced, or a simpler solution is found. Therefore, re-estimation should not only be seen as a correction of an incorrect initial estimate, but also as a response to uncertainty and changing information. However, re-estimation is not only a technical update of a numerical value. It is also an operational decision. In the case of maintenance work, where effort estimates may be connected to client approval or budget expectations, re-estimation can determine whether additional work needs to be discussed with the client or confirmed before implementation continues. Therefore, from a business perspective, re-estimation depends not only on whether the revised estimate is more accurate, but also on whether it occurs early enough.

Psychological factors can also influence re-estimation. Although more information may be available later in the task life cycle, revised estimates may still be affected by the initial estimate. Anchoring can cause the original estimate to remain a reference point, even when new evidence suggests that the task requires more effort than expected [16, 20]. Similarly, optimism and overconfidence can lead teams to delay increasing an estimate because they expect that the remaining work can still be completed within the original effort prediction [13]. As a result, re-estimation may sometimes happen later than would be useful from a planning or client communication perspective.

This makes the timing of re-estimation important. If a task is re-estimated before the active budget, based on the earlier estimate has been consumed, the revised estimate can support more proactive planning, capacity adjustment, and client communication. If the re-estimation occurs after the cumulative time spent has already exceeded the active estimate, it may still improve the accuracy of the documented estimate, but it provides less value as an early management intervention. In such cases, re-estimation functions more as a corrective or administrative update than as a proactive control mechanism.

Additionally, estimates can influence project behavior rather than only describe expected effort. Jørgensen and Sjøberg show that effort estimates may become part of the project context itself, as teams can adapt their work to the available effort allowance [14]. This means that estimates and re-estimates should not be interpreted only as neutral predictions. They can also shape how work is planned, prioritized, and executed. For this reason, this thesis treats estimation data not only as numerical project data, but also as a record of organizational decision-making under uncertainty.

3 Related Work

Existing research on software effort estimation has mainly focused on predicting the final effort, cost, size, or duration of software work before implementation begins. Previous research has proposed several approaches for effort estimation, including Bayesian network models [7], algorithmic and parametric approaches such as COCOMO [21, 15, 6], machine-learning approaches [22], and hybrid models that combine multiple estimation techniques [5]. Together, these studies show that historical project data can support and guide effort estimation practices when the available data is sufficiently complete, consistent, and representative [2, 10]. However, these approaches mainly focus on predicting effort before or at the start of implementation. Therefore, they provide less insight into how effort-related data can be used during execution, when work is already in progress and the currently approved estimate may become insufficient.

Several studies have also applied machine-learning techniques specifically to task-level software estimation. Sousa et al. [22] evaluate multiple machine-learning algorithms for estimating the effort and duration of individual software tasks across project-management datasets. Their findings demonstrate that task-level data from project management systems can contain useful estimation-related patterns. This is relevant, as it supports the idea that operational task data, such as planned effort and recorded effort, can be used for estimation-related analysis. However, their study focuses on predicting task effort or duration, rather than on detecting whether an ongoing task is at risk of exceeding the estimate that is currently active. Similarly, classification-based approaches have also been used in software project prediction. Cerpa et al. [4] use Logistic Regression to estimate software project outcomes based on data from multiple organizations, showing that Logistic Regression can be applied to software project outcome prediction. This is relevant because overrun-risk detection can also be formulated as a binary classification problem: at a specific point in a task history, a task is either at risk or not at risk of later exceeding its active estimate. Logistic Regression is therefore suitable as an interpretable baseline model for such tasks.

A more directly related work to this study is the recent work by Catana and Florescu [3], who propose an integrated framework combining machine-learning-based effort prediction with threshold-based early risk detection. Their approach predicts task duration from task-level features such as planned effort, complexity, and team experience, and then flags tasks where predicted effort deviates from planned effort beyond a defined threshold. Their study demonstrates that predictive signals from task-level data can be translated into early-warning indicators during project execution. However, such approaches often depend on contextual features that are not always systematically recorded in real-world project management systems. In many operational maintenance environments, features such as task complexity, developer experience, or implementation context may be incomplete, inconsistently documented, or unavailable for historical analysis.

This creates an important gap to study. Existing estimation models show that historical data

can support effort prediction, and recent early-risk approaches show that task-level signals can be used for warning mechanisms. However, less attention has been given to monitoring approaches based only on routinely available effort-consumption data during task execution. In client-facing maintenance work, this distinction matters because the active estimate is not only a prediction of effort, but also represents the currently approved effort budget for a task. When the amount of work hours approaches or exceeds the active estimate, the issue is no longer only estimation accuracy, but also planning, re-estimation, and client communication.

Therefore in this study, we address the gap by shifting the focus from predicting final task duration to monitoring effort consumption against the estimate that is currently active. Since features such as task complexity and team experience are frequently absent or inconsistently documented in operational project management systems, this study is limited to structured effort-consumption data that is already routinely recorded in the case-study environment. In client-facing maintenance work, the active estimate represents the approved effort budget for a task, and this study therefore examines whether historical version-level records can be used to identify early when ongoing work is approaching or likely to exceed it. Logistic Regression is used as an interpretable baseline model to evaluate whether the available project-management data contains useful signals for review-based monitoring.

4 Case Study Environment

This thesis is conducted as a graduation internship at Netcompany Netherlands B.V. Netcompany is an IT consultancy company that delivers digital solutions and services to public- and private-sector clients. The internship takes place within the Application Project/Services department, which is responsible for supporting, maintaining, and improving software systems after deployment. The APS department provides a relevant environment for studying effort estimation because much of its work consists of task-level maintenance activities, such as client-requested changes, service requests, bug fixes, and other improvements to existing systems. This requires the team to constantly estimate the expected effort of individual work items and to document their progress during implementation. Therefore the cumulated work item data can be used to examine how effort estimates compare to recorded effort, how estimates are updated over time, and whether effort-consumption patterns indicate potential estimate overruns.

The data used in this study comes from Netcompany's internal work management platform, Toolkit, where work items are registered as *cases*. Cases contain documented information such as the case identifier, title, case type, life cycle status, responsible team member, effort estimate, cumulative time spent, version number, and timestamps. Although Toolkit contains multiple case types, this thesis focuses specifically on *Change* cases. Change cases are client-requested modifications to an existing system, such as changes to functionality, infrastructure, or other system features. These cases are relevant for this study, as they require formal effort estimation and are directly connected to planning, budgeting, and client approval. When a case requires more effort than was initially estimated and communicated to the client, additional confirmation is required before the extra budget can be approved and implementation can continue. This makes early detection of potential effort overruns necessary and valuable, as it could help signal to teams the need for re-estimation and client communication before the approved estimate has already been exceeded in practice.

Figure 2 illustrates the estimation process flow in the case study environment. A Change case begins with a client request, after which the team conducts a requirements analysis and proposes a solution. A build and release estimate is then agreed upon through Planning Poker[1], and submitted for client budget approval. Once approved, the active estimate is set and development, testing, and time registration begins. If the team or a developer determines that the approved estimate is no longer sufficient, the case enters a re-estimation loop, where a request for additional budget is prepared, including an explanation of the remaining work, and submitted for renewed client approval. Once approval is received, the active estimate is updated in the system. This loop is central to the study because it shows that active-estimate overruns are budget-control events, where the timing of overrun recognition matters: if the need for re-estimation is identified too late, teams have less time to adjust planning, prepare client communication, or obtain approval before the approved effort budget is consumed, and might have to put work on hold or reduce quality of the deliverable.

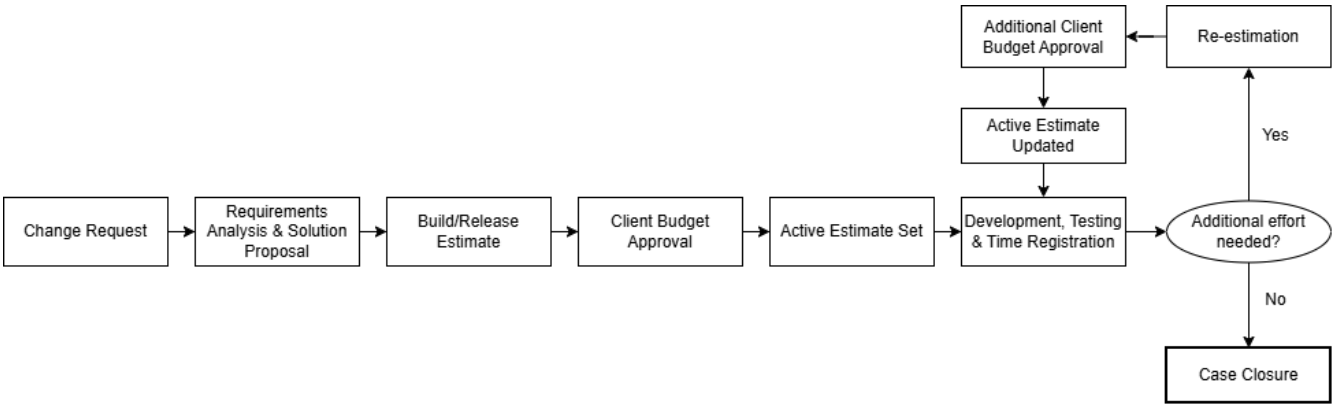


Figure 2: Estimation process flow in the case study environment

5 Methodology

In this section we provide an overview of the research design, data collection, preprocessing, and the analytical approach used to answer the research questions. Section 5.1 outlines the overall study design. Sections 5.2 - 5.6 cover the operational definitions, data collection, preprocessing, and case-study analysis methods. Section 5.7 describes the model development approach, including feature engineering, evaluation metrics, and the consumption-ratio threshold comparison. Section 5.8 outlines the practitioner validation interview approach, and Section 5.9 describes how we combine quantitative and qualitative findings into practical recommendations for the organization.

5.1 Research Design

This study follows an empirical mixed-methods case-study design, split into 3 parts, combining quantitative analysis of historical task data with exploratory predictive modeling and confirmatory

[1]Planning Poker is an estimation technique in which team members independently in a group setting estimate the required effort for a task using numbered cards. The estimates are then discussed in iterative rounds until a shared consensus is reached.

practitioner interviews. Table 1 provides an overview of the methodological steps we take and follow throughout the study.

The first phase is a descriptive-analytical case-study analysis. Using reconstructed version-level task histories, this phase examines how accurately initial estimates reflect final recorded effort, how frequently active estimates are exceeded during task execution, and at what level of effort consumption formal estimate updates are documented in the system. This phase addresses RQ1, RQ2, and RQ3, and serves as the empirical grounding for the rest of the study by characterizing the estimation and re-estimation behavior present in the case-study data, and motivating the following modeling and interview components.

The second phase is an exploratory predictive feasibility analysis addressing RQ4. A Logistic Regression classifier is trained on version-level snapshots of case histories to assess whether routinely collected effort-consumption data contains a detectable signal for overrun risk. The purpose of this component is not to develop a production-ready prediction model, but to establish whether the available data is informative enough to support structured monitoring. Model performance is compared against simple consumption-ratio threshold rules to determine whether the predictive signal can be captured without a trained classifier, and to identify the most practical form of an early-warning mechanism given the available data.

The third phase consists of exploratory confirmatory semi-structured practitioner interviews, conducted after the quantitative results are found. Two hypotheses derived from the quantitative findings are presented to practitioners with direct experience in the case-study environment, to validate whether the observed patterns reflect operational reality and to assess the feasibility of introducing consumption-based monitoring into the existing workflow.

Table 1: Overview of the methodological flow and related research questions

Step	**Activity**	**Method**	**RQ**
1	Data collection and preprocessing	Filter the raw dataset export, remove duplicate and incomplete records, select completed Change cases, and reconstruct version histories chronologically.	–
2	Construction of analysis views	Transform the cleaned case histories into a case-level dataset and a version-level dataset.	–
3	Initial estimation accuracy analysis	Compare the initial formal estimate with the final cumulative time spent and classify cases as underestimated, overestimated, or accurate. Apply accuracy buffers as a sensitivity check.	RQ1
4	Active-estimate overrun analysis	Segment case histories into active-estimate periods and identify whether the active estimate is exceeded before the next re-estimation or case closure.	RQ2
5	Operational overrun scale analysis	Calculate the total amount of effort spent beyond active estimates across all overrun cases.	RQ2
6	Re-estimation frequency and timing analysis	Count re-estimations per case and classify re-estimation timing using effort-consumption ratio thresholds.	RQ3
7	Logistic Regression model analysis	Train and evaluate baseline, reduced, and extended Logistic Regression models using grouped five-fold cross-validation.	RQ4

Table 1: Overview of the methodological flow and related research questions (continued)

Step	Activity	Method	RQ
8	Feature contribution analysis	Interpret the Logistic Regression model using coefficient-based feature analysis to identify which variables contribute most strongly to overrun-risk classification.	RQ4
9	Consumption-ratio threshold analysis	Compare the Logistic Regression model with simple threshold rules based on consumption ratios of 0.50, 0.75, 0.80, 0.85, and 0.90.	RQ4
10	Practitioner validation interviews	Conduct interviews with practitioners to contextualize findings, validate observed patterns, and assess practical feasibility.	–
11	Derivation of practical recommendations	Combine quantitative findings, and practitioner input to derive practical workflow improvement recommendations.	–

5.2 Operational Definitions

To analyze estimation behavior consistently, this study follows a set of definitions. These definitions specify how effort estimates, re-estimations, estimation accuracy, estimate overruns, and effort consumption are measured throughout the analysis.

Key Operational Definitions

- **Case:** A single work item registered in the project management system, representing a unit of work to be estimated, planned, and executed.
- **Effort estimate:** The predicted amount of effort, expressed in hours, required to analyze, develop, test, implement and release a task.
- **Version entry:** A single recorded update to a case in the project management system, created whenever a case attribute is updated. Each version entry captures the state of a case at a specific point in time, including the current effort estimate and cumulative time spent at that moment. A case history consists of all version entries belonging to the same case.
- **Initial formal estimate:** The first meaningful effort estimate assigned to a case. It is determined by the first nonzero effort estimate recorded in a case history, since some cases initially contain a placeholder value of zero before a formal estimate is assigned. In such cases, the first later nonzero value is treated as the initial formal estimate. The transition from zero to a nonzero estimate is therefore interpreted as the initial estimation and is not counted as a re-estimation.
- **Re-estimation:** A system-recorded revision of the effort estimate during the case life cycle. It is defined as any change from one nonzero effort estimate to another nonzero

effort estimate within the same case history after the initial formal estimate has been assigned. This means that only estimate changes occurring after the initial formal estimate are counted as re-estimations.

- **Active estimate:** The effort estimate currently valid for a case at a given point in its life cycle. Once a nonzero estimate is assigned, this value remains the active estimate until it is replaced by a later nonzero estimate. The active estimate is used as the reference point for comparing recorded effort consumption.
- **Cumulative time spent:** The total effort logged on a case up to a specific point in its life cycle. It is recorded progressively across version entries, meaning that each version entry captures the running total of all effort logged on the case up to that moment, rather than only the effort added since the previous update.
- **Estimation accuracy:** A measure of how closely the initial formal estimate reflects the actual effort required. It is determined by comparing the initial formal estimate with the final cumulative time spent. A case is classified as *underestimated* when the final cumulative time spent is higher than the initial formal estimate, *overestimated* when the final cumulative time spent is lower than the initial formal estimate, and *accurately estimated* when both values are equal.
- **Estimate overrun:** A situation in which the cumulative time spent on a case exceeds the active effort estimate before it is revised or the case is completed. At the version level, this is determined by comparing the cumulative time spent recorded at each version entry with the active estimate at that same point in the case history.
- **Effort-consumption ratio:** A measure of how much of the active estimate has already been consumed at a specific point in the case life cycle. It is calculated by dividing the cumulative time spent by the active effort estimate. This ratio indicates how much of the active estimate has already been consumed at a specific point in the case life cycle. A ratio below 1 indicates that the recorded effort is still below the active estimate, and a ratio above 1 indicates that the active estimate has been exceeded, and an overrun has occurred.

5.3 Data Collection

The initial dataset contains 7493 documented task records from an ongoing software maintenance project from the case study environment including the version histories of individual work items. Each version history entry represents a documented update to a work item, which makes it possible to reconstruct how individual cases have evolved throughout their life cycle. For each case, 34 attributes are documented. These attributes include numerical fields, such as the effort estimate, cumulative time spent, and a version number that increments with each recorded update to a case attribute, as well as categorical and textual fields, such as case title, case type, life cycle status, and responsible team member.

5.4 Data Preprocessing

Before conducting the analysis, we transform the raw dataset following the steps summarized in Table 2. By preprocessing we ensure that each case has the information required to reconstruct its case history and compare estimated effort with recorded effort. The main filtering and transformation steps are based on case type, completion status, version order, effort estimate, and cumulative time spent.

First, we limit the dataset to Change cases. Although the original dataset contains multiple case types, we maintain only Change cases because they consistently contain the necessary documented effort estimates and recorded time spent for this study. After selecting the relevant case type, we remove duplicate task records from the dataset to ensure that the same case information is not counted for multiple times in the analysis. Second, we keep only completed cases, as estimation accuracy can only be evaluated when the final cumulative time spent is known. Cases that are still ongoing, canceled, or otherwise incomplete are therefore excluded from the final analysis dataset. Third, we remove cases without a valid nonzero effort estimate or without documented cumulative time spent, as these entries are required to compare estimated effort with actual recorded effort and to determine whether and when overruns occur. After these filtering steps, 1623 completed Change cases remain for analysis. The version histories of these cases are then ordered chronologically for each case to reconstruct how estimates and cumulative time spent evolve over time. Lastly, redundant version entries caused by system-generated updates that reflect no meaningful change in effort estimate or cumulative time spent are removed.

Table 2: Overview of preprocessing steps applied to the raw data export

Step	Title	Purpose	Removed Cases	Retained Cases
1	Select Change cases	Focus the analysis on the case type with consistent formal effort estimate and recorded time spent documentation.	5153	2340
2	Remove duplicate cases	Prevent repeated case information from being counted multiple times.	45	2295
3	Maintain completed cases	Ensure cases have a complete documented history with final cumulative time spent available.	263	2032
4	Remove cases without valid estimates or time spent	Retain only cases where estimated effort can be compared with recorded effort.	409	1623
5	Order version histories chronologically	Reconstruct how estimates and cumulative time spent evolve during each case life cycle.	0	1623
6	Remove redundant system updates	Exclude version rows where neither the estimate nor cumulative time spent changed.	0	1623

5.5 Construction of Case-Level and Version-Level Analysis Views

After preprocessing, we transform the cleaned case histories into two analysis views: a case-level view and a version-level view. Both are derived from the same underlying version histories, but they are used for different parts of the analysis.

The case-level view summarizes each completed Change case into one row. For each case, the version history is ordered chronologically and used to extract the initial formal estimate, final estimate, final cumulative time spent, and the number of re-estimations. We use this view to analyze initial estimation accuracy and overall re-estimation frequencies. The second version-level view keeps the chronological structure of the case histories, where each row represents a meaningful version update where the effort estimate or cumulative time spent changes. For each version, the active estimate and cumulative time spent at that point in the case life cycle are recorded. Based on these values, the effort-consumption ratio is calculated by dividing cumulative time spent by the active estimate. From this format we identify active-estimate overruns, examine when re-estimations occur in relation to effort consumption, and construct the version-level observations used for the early-warning model.

5.6 Quantitative Case Study Analysis

The analysis focuses on structured numerical fields. Textual attribute fields, such as case titles and descriptions, are not used in the quantitative analysis or model development because they are often inconsistently documented and would require additional text-processing methods. We use categorical attributes only where needed for grouping, and ordering the case histories.

5.6.1 Initial Estimation Accuracy

RQ1: *How accurate are early task-level effort estimates compared to the actual effort recorded, and how does estimation accuracy vary?*

Previous research suggests that early estimates often differ from the true effort required [12, 18]. Therefore, we examine RQ1 as the grounding step to determine early estimation accuracy of initial estimates and whether effort overruns are common in the case-study environment.

To answer RQ1, the first formal estimate of each completed Change case is compared with the documented cumulative time spent at completion. Based on this comparison, each case is classified as accurate, overestimated, or underestimated. This classification makes it possible to identify the dominant estimation-error pattern in the case-study data and to examine whether it aligns with previous research, which identifies underestimation as a common issue in software effort estimation practice [12, 23]. Because effort estimates are mostly documented as rounded whole-hour values, while time spent may be logged in smaller increments, strict equality between estimated and recorded effort may be too restrictive. Therefore, we conduct a sensitivity check using accuracy buffers of 0.25 hours and 2 hours. These buffers are used to examine whether cases classified as inaccurate under strict equality are only minor deviations from the estimate, or whether estimation errors remain even when small error is allowed.

5.6.2 Active-estimate overrun analysis

RQ2: *How often do active-estimate overruns occur during the case life cycle, and how large are these overruns?*

To answer RQ2, the analysis shifts from the case-level view used in RQ1 to a version-level analysis of active estimates. RQ1 compares the first formal estimate with the final cumulative time spent, but this only evaluates the initial estimate. Since estimates can be revised during the case life cycle, the active estimate does not always remain constant throughout the case. Therefore, RQ2 examines whether the estimate that is active at a given point in the case history is exceeded before it is revised or before the case is completed.

For this analysis, we divide each case history into active-estimate periods. An active-estimate period starts when a nonzero estimate becomes active and lasts until that estimate is revised or until the case is completed. For each active-estimate period, the active estimate is compared with the maximum cumulative time spent recorded within that same period. If the cumulative time spent exceeds the active estimate before the next re-estimation or case closure, the period is classified as an active-estimate overrun.

The active-estimate overrun analysis is reported from three perspectives. First, we identify the number of completed Change cases in which at least one active estimate is exceeded during the case life cycle. This provides a case-level overview of how frequently active-estimate overruns occur in the dataset. Second, we assess the size of these overruns by measuring the total exceeded amount of effort recorded and expressing it as a percentage of the active estimate. These percentages are then grouped into severity categories to distinguish minor deviations from larger overruns. Third, we combine all overrun hours and express the total as both an absolute value and as a percentage of the total effort logged across all completed Change cases, to provide a practical indication of the scale of unbudgeted effort relative to the overall project workload.

5.6.3 Re-estimation frequency and timing

RQ3: How often are effort estimates formally updated in the system, and at what level of effort consumption do these updates occur?

After determining whether active estimates are exceeded in RQ2, we further analyze how often effort estimates are formally updated during the case life cycle and at what level of effort consumption these updates appear in the project-management system. In this study, re-estimation is measured as a recorded change from one nonzero effort estimate to another nonzero effort estimate after the initial formal estimate has been assigned. Therefore, the analysis captures the timing of recorded estimate updates in the system, rather than necessarily capturing the exact moment at which developers or teams first recognized that additional effort might be needed. This distinction is important because the available dataset only shows when an estimate value changes in the system. It does not directly show why the estimate was changed, who initiated the change, whether the team had already identified the need for additional effort earlier, or whether the update followed an approval process outside the system. Therefore, re-estimation timing is not interpreted as a direct measure of developer awareness or decision timing.

First, we identify how many cases contain at least one recorded re-estimation. We then count

the number of re-estimations per case to determine how common formal estimate updates are during case life cycles and whether cases are usually updated once or multiple times. Next, we group recorded re-estimations into timing categories based on the effort-consumption ratio immediately before the estimate update. This allows us to determine whether formal estimate updates are recorded early, near the active estimate limit, or only after the active estimate has already been exceeded. This means that recorded re-estimation timing is measured by effort consumption rather than by calendar time. Since the available data does not capture the internal moment at which overrun risk was first recognized, it is not possible to calculate how long teams may have been aware of the need for additional effort before the formal estimate update was entered into the system. Similarly, because the version-history timestamps are not reliable enough for this purpose, it is not possible to calculate how many calendar days cases remain in an overrun state before formal re-estimation. A future extension would therefore be to record a separate internal review or overrun-risk recognition timestamp, in addition to the later formal estimate update.

5.7 Model Development

After analyzing estimation accuracy, active-estimate overruns, and documented estimate-update patterns, this study uses the version-level case histories to explore whether historical effort-consumption data can support review-based overrun monitoring. The purpose of the model is not to predict the final effort required for a case, as the available data does not contain a timestamp for internal risk awareness, the start of a re-estimation review, or the moment a re-estimation request was submitted for client approval. Instead, the model is used to assess whether the current recorded state of a case contains signals that the active estimate will later be exceeded in the version history before the next formal estimate update or case closure.

RQ4: *To what extent can historical effort-consumption data support classification of ongoing cases at risk of exceeding their active effort estimate?*

5.7.1 Target Variable

The target variable in the model indicates whether a version-level snapshot is at risk of an active-estimate overrun. Since the aim of the model is to support early warning during case execution, the label is based on whether the active estimate at a given snapshot is exceeded later within the same active-estimate period. A label of 1 is assigned when the cumulative time spent later exceeds the active estimate within the same active-estimate period. This indicates that the estimate became insufficient before it was replaced or before the case was completed. A label of 0 is assigned when the cumulative time spent does not exceed the active estimate during that period. This target should be interpreted as a recorded overrun-risk label, not as a direct measure of developer awareness or re-estimation decision timing. In the case-study workflow, a formal estimate update may occur only after internal review, preparation of a remaining-work breakdown, and client approval. Therefore, the model does not learn when overrun risk was first noticed or when re-estimation was internally triggered. It only learns whether the information available at a given version-level snapshot is associated with later exceedance of the currently active estimate in the system record.

5.7.2 Feature Engineering and Feature Set Comparison

The model input features are extracted from the available version-history data. Since the model is intended to support overrun-risk detection for ongoing tasks, only information that would be available at the current version snapshot is included. This means that the model does not use future information, such as the final recorded effort of the case, as an input feature.

We first create a baseline feature set. This set contains the five most direct indicators of effort consumption and re-estimation behavior for each task. It includes the active effort estimate, the cumulative logged time spent, the number of re-estimations occurred so far, the effort consumption ratio, and whether the current version row represents a re-estimation in itself. These features describe the current state of the task, by indicating how much effort is currently estimated, how much effort has already been consumed, how close the case is to reaching its active estimate, and whether the estimate has already been adjusted during the case life cycle. We use the baseline model to test whether simple information from the version history is sufficient to identify cases that may be at risk of exceeding their active estimate. We consider the consumption ratio to be an especially important feature because it directly captures the relationship between the active estimate and the effort already spent. We extract this feature by dividing cumulative logged time spent by the active estimate. A higher consumption ratio means that a larger share of the current estimate has already been consumed. For example, a ratio close to 1 indicates that the case is approaching its active estimate, while a ratio above 1 means that the active estimate has already been exceeded.

After training the baseline model, we conduct additional feature-set comparisons to examine how model performance changes with access to different types of information. First, we test a reduced feature set by removing the effort consumption ratio from the baseline model. We remove this feature because it is expected to the most direct warning signal for overrun risk, as it captures the relationship between the active estimate and the cumulative effort already spent. Second, we test an extended feature set by adding additional historical version-history variables. These include the initial formal estimate, remaining effort amount, version update number indicator, time-spent increase since the previous version update, and whether the case has already been re-estimated. These variables provide more context about how the case has developed up to the current update. The extended model therefore tests whether earlier case-history information adds useful predictive value beyond the current estimate, logged time spent, consumption ratio, and re-estimation behavior included in the baseline model.

By comparing the baseline, extended, and reduced feature sets, we evaluate whether overrun risk can be detected from the available version-history data, but also which types of information contribute most to model performance. From a practical perspective, this comparison helps determine whether an early-warning approach would require more detailed historical feature engineering, or whether simpler operational indicators already capture most of the useful warning signal.

Table 3: Comparison of Feature Sets Used for Model Testing

Feature	Baseline Model	Reduced Model	Extended Model
Active effort estimate	✓	✓	✓
Cumulative logged time spent	✓	✓	✓
Re-estimation count	✓	✓	✓
Re-estimation event indicator	✓	✓	✓
Effort consumption ratio	✓	–	✓
Initial formal estimate	–	–	✓
Remaining effort amount	–	–	✓
Version number indicator	–	–	✓
Time-spent increase since previous version update	–	–	✓
Overrun label	Target	Target	Target

5.7.3 Logistic Regression Model Development

We select Logistic Regression (LR) as the baseline model for this study, as the task is a binary classification problem, where each version-level snapshot is classified as either at risk or not at risk of exceeding its active effort estimate. With LR we can assign an overrun-risk score for each version-level snapshot, and represent the likelihood that the active effort estimate will later be exceeded in the recorded version history before the next formal estimate update or case closure. This choice is also supported by previous software estimation and prediction research, where Logistic Regression has been used to classify or predict software project and task outcomes [3, 22]. In addition, LR is suitable for this exploratory study because it is interpretable. By examining the model coefficients and odds ratios, the contribution of each input feature can be assessed to determine whether the selected features provide meaningful warning signals for overrun-risk detection.

The preprocessing and classification steps are implemented as a pipeline. First, although missing values are not expected after the earlier data preprocessing steps, missing numerical values are first filled in using the median, due to outliers. The numerical features are then scaled. Because the model dataset is constructed at the version level, multiple rows can belong to the same case. These rows are not independent observations, because they represent different snapshots from the same case history. To prevent data leakage, we apply five-fold grouped cross-validation using the case identifier as the grouping variable. In each iteration, four folds are used for training and one fold is left out for testing. This ensures that all version entries belonging to the same case are assigned either to the training set or to the test set, but never to both.

5.7.4 Evaluation Metrics

For each fold in the five-fold grouped cross-validation, a confusion matrix is calculated. The structure of a confusion matrix is shown in Figure 3. Predictions from all five test folds are then combined into one aggregated confusion matrix. This provides an overall view of the types of errors made by the model. For the other evaluation metrics, performance is calculated separately for each fold and then reported as the mean and standard deviation across the five folds.

In this study, the positive class consists of case-version observations where the active estimate is later exceeded. The negative class consists of observations where the active estimate is not later exceeded. The evaluation therefore focuses on whether the model can identify moments in a task life cycle that contain warning signals for a potential estimate overrun. In this context, correctly identifying observations that are actually at risk of exceeding their active estimate is crucial. Therefore, recall, also referred to as sensitivity, is a central metric considered in this study. A false negative means that an observation with actual overrun risk is not flagged by the model, which may leave a potential overrun unnoticed. In contrast, a false positive may lead to unnecessary review, but this is less problematic than missing a real risk signal.

Precision is still relevant, because too many false warnings would reduce the practical usefulness of the model. However, a model with high precision but low recall would not be suitable for early-warning support, because it would miss many actual overrun-risk observations. For this reason, the F1-score is also reported, as it combines precision and recall into a single measure. Accuracy is included for completeness, but is not equally relevant, as it treats false positives and false negatives as equally important, while their practical consequences differ in an early-warning context.

The entries of a confusion matrix are as follows:

- **True Positive (TP):** A case-version observation is correctly classified as at risk. The model predicts that the active estimate will later be exceeded, and the recorded case history confirms that the active estimate gets exceeded before the next formal estimate update or closure.
- **False Positive (FP):** A case-version observation is incorrectly classified as at risk. The model predicts that the active estimate will later be exceeded, but the recorded case history shows that the active estimate is not exceeded before the next formal estimate update or closure. In practice, this would correspond to an unnecessary review signal.
- **True Negative (TN):** A case-version observation is correctly classified as not at risk. This means that the model predicts that the active estimate will not be exceeded, and the recorded case history confirms that the active estimate does not get exceeded until the next formal estimate update or closure.
- **False Negative (FN):** A case-version observation is incorrectly classified as not at risk. This means that the model predicts no overrun risk, but the recorded case history later shows that the active estimate is exceeded before the next formal estimate update or closure. In practice, this would correspond to a missed point for an earlier review signal.

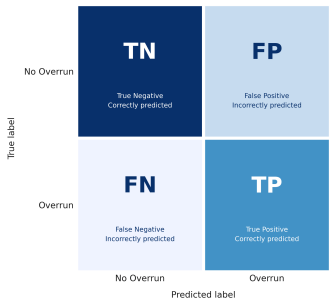


Figure 3: Confusion Matrix

Based on the fold-level confusion matrix results, we report on the mean and standard deviation of the following metrics across the five folds:

- **Accuracy** measures how many observations are correctly classified by the model out of all evaluated observations. A high accuracy means that the model correctly classifies many observations overall, while a low accuracy means that many observations are misclassified. However, accuracy does not show whether the errors are false positives or false negatives, however in this context the distinction is important because missing an actual overrun risk is more problematic than flagging a case unnecessarily.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- **Precision** measures how many of the classified as at risk are observations where the active estimate is later exceeded. A high precision means that most warnings produced by the model are correct. A low precision means that many observations are incorrectly flagged as at risk, which may lead to unnecessary review.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

- **Recall**, also referred to as sensitivity, measures how many of the observations where the active estimate is later exceeded are correctly identified by the model. A high recall means that the model detects many of the actual overrun-risk observations. A low recall means that most actual at risk cases go unnoticed by the model, which is undesirable for an early-warning model.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

- The **F1-score** combines precision and recall into a single balanced metric. A high F1-score indicates that the model is able to detect actual risk cases while also limiting the number of false warnings.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

Lastly, we use the ROC-AUC score to evaluate how well the model distinguishes between observations where the active estimate is later exceeded and observations where it is not exceeded. A ROC-AUC value of 0.5 indicates performance equivalent to random guessing, while higher values indicate stronger discrimination between the two classes. A higher ROC-AUC means that observations with actual overrun risk are more likely to receive higher predicted risk scores than observations without overrun risk.

5.7.5 Consumption-Ratio Threshold Comparison

In addition we compare the best-performing model with a threshold-based warning approach based on the effort-consumption ratio. We include this to assess whether overrun-risk monitoring could also be supported by a simple, transparent rule based only on the relationship between the active estimate and cumulative time spent.

For this approach, we apply the threshold rule to all version-level snapshots in the model dataset. Each snapshot represents the state of a case at a specific point while it was still in progress. A snapshot is classified as at risk when its consumption ratio is equal to or higher than the selected threshold, and as not at risk when it is below the threshold. We test five thresholds: 0.50, 0.75, 0.80, 0.85 and 0.90. The 0.50 threshold represents an early warning point, where half of the active estimate has been consumed. The 0.75 threshold represents a review point, where most of the active estimate has been consumed but the estimate has not yet been exceeded. The 0.90 threshold represents an urgent review point, where the case is close to consuming the full active estimate, and the thresholds in between function as exploratory thresholds for determining the best performing option. We then evaluate each threshold rule using the same metrics as the Logistic Regression model and compare their performance, and assess whether a simple consumption-ratio rule could be valuable enough for the organization to implement.

5.8 Practitioner Validation Interviews

To ground the quantitative findings of RQ1-RQ4 in practice, we conduct two confirmatory semi-structured interviews with a senior software engineer, and a team lead/Scrum Master from the case-study project. Both practitioners have direct experience with Change cases, effort estimation, re-estimation, and the workflow surrounding client communication and budget approval.

By following a semi-structured format, we structure the interviews around three main core topics, with follow-up questions being adapted to the practitioner's role and experience. First, we determine the practical role of effort estimates in the case-study environment, including what estimates represent, how they are communicated to the client, and what happens when additional effort is required. Second, we validate and interpret the quantitative patterns identified in the data, particularly the frequency of active-estimate overruns and the timing of re-estimations. This includes discussing whether late re-estimation records reflect late awareness by the team, or whether they result from delays from the client approval side. Third, the interviews explored current practices and the practical feasibility and benefits of integrating consumption-based monitoring, including which consumption review triggers would would be useful, how such signals should be implemented in the existing workflow, and what conditions would be necessary for reliable adoption.

Before conducting the interviews we derive two hypotheses from the quantitative results from Sections 6.1 and 6.2, which we later rate as confirmed, partially confirmed or not confirmed based on our interview findings:

- **H1**: Late re-estimation is a consequence of limited visibility and monitoring of effort consumption during task execution, rather than estimation inaccuracy alone.
- **H2**: A structured, consumption-ratio-based review trigger would be perceived by practitioners as a feasible and valuable improvement over the current practice.

5.9 Derivation of Practical Recommendations

By combining the results from the quantitative case-study analysis, the early-warning model, threshold comparison, and the practitioner validation interviews, we derive a set of practical recommendations for the organization to implement. The quantitative analysis identifies the scale and timing of estimation and re-estimation patterns, the model comparison assesses whether existing effort-consumption data contains a usable warning signal, and the practitioner interviews provide operational context on workflow fit, feasibility, and adoption conditions. Together, we combine these sources to derive practical process improvements for increased and earlier overrun awareness during task execution.

6 Results and Discussion

In this section we present, and discuss the results of the study across four research questions. Section 6.1 reports on the case-level estimation accuracy, active-estimate overrun frequency, and re-estimation timing analysis addressing RQ1, RQ2, and RQ3. Section 6.2 presents the early-warning model evaluation and threshold comparison for RQ4, followed by the practitioner validation interviews and practical recommendations in Sections 6.3 and 6.4.

6.1 Case-level Estimation and Overrun Analysis

6.1.1 Data Availability and Case History Characteristics

The raw dataset contains 7493 cases across multiple case types, including Changes, Events, Incidents, and Service Requests. The distribution of case types in the export is presented in Figure 4. During data exploration, we find that data completeness varies across case types. In the full dataset, 2457 cases contain no documented cumulative time spent, and 3251 cases never receive an effort estimate. Most of these incomplete cases belong to non-Change case types, which is consistent with the organizational procedure where formal effort estimation is required only for Change cases. For this reason, we focus our analysis specifically on Change cases, as they most consistently contain both formal estimates and logged time spent. After applying the preprocessing criteria described in Section 5.4, we are left with 1623 completed Change cases for analysis.

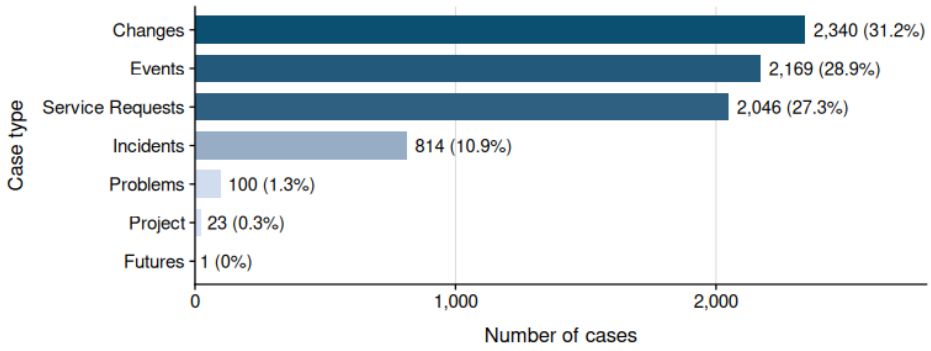


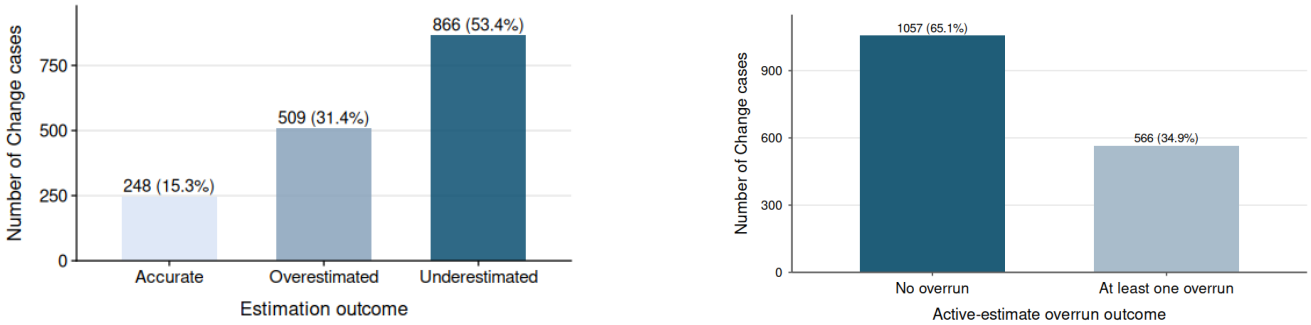
Figure 4: Distribution of case types in the raw dataset export

An important data quality observation we make is the mismatch in precision between effort estimates and logged time spent. Effort estimates are mainly recorded as rounded whole-hour values, while cumulative time spent is often logged in smaller increments, typically quarter or half hours. This difference in precision is important when interpreting exact estimation accuracy, especially for cases classified as accurately estimated. Additionally, discussions with developers indicate that time registration may sometimes be rounded or adjusted in practice, meaning that the documented time spent does not always precisely correspond to the actual effort invested in reality, which is an important consideration for further analysis.

The data exploration also reveals a practical limitation. The usable dataset consists of 1623 cases with valid estimates and logged time, and represents only a subset of the full export. An early-warning system based on consumption ratios could only be applied to cases where both the active estimate and cumulative time spent are actively and consistently documented. Therefore, incomplete documentation would limit the applicability and reduce the reliability of any automated monitoring approach.

6.1.2 RQ1 and RQ2: Estimation Accuracy and Active-Estimate Overruns

Results. As shown in Figure 5a, across the 1623 completed *Change* cases, only 15.3% are accurately estimated under strict equality between the initial formal estimate and final cumulative time spent. The majority of cases, accounting for 53.4% of the dataset, are underestimated, while 31.4% are overestimated. As a sensitivity check we also apply buffers of ±0.25 hours and ±2 hours. The smaller buffer leads to only a minor change in the amount of accurately estimated cases, increasing from 15.3% to 16.6%, while the wider ±2 hour buffer, increases the share of accurately estimated cases to 38.5%. This shows that even with a wider buffer more than 60% of cases remain inaccurately estimated, suggesting that inaccuracy therefore cannot purely be explained by rounding or minor time-registration differences. This shows that underestimation is the most common outcome among early estimations, which is consistent with previous research identifying it as a commonly recurring issue in software effort estimation practice [23, 8, 18].



(a) Initial estimation accuracy

(b) Active-estimate overrun frequency

Figure 5: Overview of initial estimation accuracy and active-estimate overrun frequency in completed Change cases

Because estimates can be revised during task execution, initial underestimation does not always result in overruns. However, active-estimate overruns still occur in 34.9% of cases, as shown in

Figure 5b, corresponding to more than one third of all cases. The severity analysis, depicted in Figure 6, further shows that most overruns fall within the 0-25% overrun range, which is consistent with the most commonly reported error ranges reported in previous literature [12, 23]. However, a minority of cases exceed their active estimate by 100% or more, indicating that overruns are not always limited to small errors. As reported in Table 4, across the five-year project, 7602 hours were logged beyond active estimates, corresponding to 14.4% of all recorded effort across the analyzed tasks. In other words, the project required approximately 316.6 additional full working days of effort beyond what was expected.

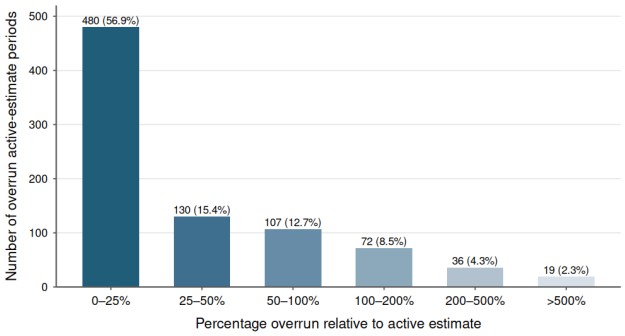


Figure 6: Distribution of active-estimate overrun severity by percentage relative to the active estimate

Table 4: Scale of Active-Estimate Overrun Effort

Measure	Value
Share of total logged effort	14.4%
Total logged effort	52870 h
Active-estimate overrun effort	7602 h
Equivalent person-days	316.6

Discussion. The 53.4% underestimation rate is consistent with previous findings from effort estimation literature [12, 8, 3]. This is relevant, as the analyzed work consists of maintenance Change cases, where the required effort often depends on existing system behavior, hidden dependencies, and implementation limitations that may only become clear during execution. Therefore, the high underestimation rate should not be interpreted only as poor estimation practice, but also as evidence that task-level maintenance estimates are made under high uncertainty. This aligns with the cone of uncertainty, which suggests that early estimates become more reliable only as more information about the task and system context becomes available [7].

The more organizationally significant finding is the scale of active-estimate overruns. Across the five-year period, 7602 hours were logged beyond active estimates, corresponding to approximately 317 additional working days of unbudgeted effort. From a client perspective, this translates directly into unplanned cost exposure. In time-and-materials arrangements, this can lead to invoices that substantially exceed the originally agreed expectations, reducing cost predictability and potentially reducing trust. In fixed-price projects, the same volume of overrun effort affects internal processes, shrinking margins and increasing pressure to compensate through scope reductions or quality trade-offs. In both cases, the outcome is a budget-control problem that becomes visible only after it has already occurred. The observed 14.4% active-estimate overrun rate therefore reflects not only estimation inaccuracy, but also the absence of a structured feedback mechanism that flags and addresses escalating effort consumption during execution.

6.1.3 RQ3: Re-estimation Frequency, and Timing

Results. Re-estimation appears to be a common procedure, with 44.9% of all cases being revised at least once during their life cycle. The remaining 55.1% of cases, corresponding to 895 out of the 1623 completed Change cases, never received re-estimations and were completed with their initial formal estimate unchanged. However, most re-estimated cases are revised only a limited number of times. Figure 7 shows that almost half, 49.7%, of re-estimated cases are revised once, while higher re-estimation counts occur less frequently. Re-estimation frequency appears to be related to task size. Figure 8 shows that cases with higher re-estimation counts have higher final cumulative time spent. This suggests that larger cases are more likely to require estimate updates, which is consistent with the expectation that larger work items contain more uncertainty and complexity [18, 8]. To distinguish this from the size expected at the start of the case, we also examined the relationship between re-estimation frequency and the initial formal estimate. This comparison did not show the same increasing pattern: cases with more re-estimations were generally not initially estimated as large, and instead often started from relatively small initial estimates. This suggests that repeated re-estimation is more likely driven by effort growth and growing complexity during execution than by cases being conceived as large from the beginning.

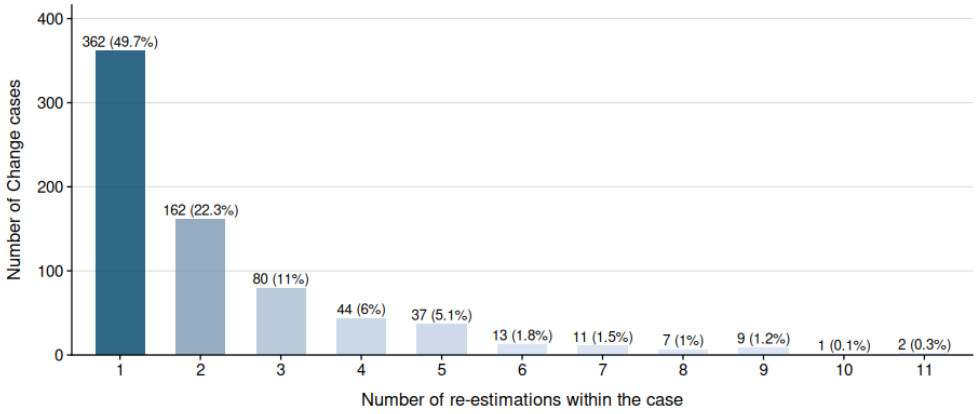


Figure 7: Distribution of re-estimation counts per completed Change case

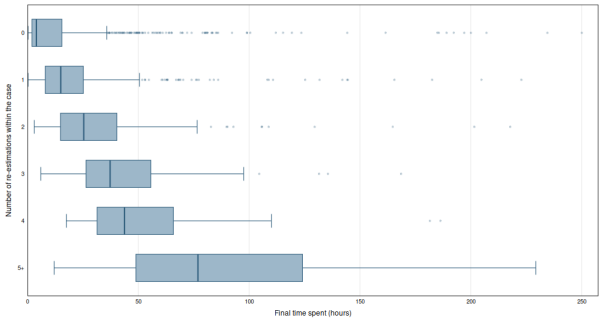


Figure 8: Final cumulative time spent by re-estimation count

A closer examination of documented estimate-update timings reveals a limitation in how re-estimations become visible in the system. As shown in Table 5, 37% of documented estimate

updates are recorded after the active estimate has already been exceeded (100-150%), and 9.8% are recorded at consumption ratios above 150%. Only 28.3% of documented estimate updates are recorded before 75% of the active estimate has been consumed. These results show when revised estimates become visible in the version history, but they should not be interpreted as the exact moment at which developers or teams first became aware that additional effort was needed. In the case-study workflow, a formal estimate update may occur only after internal review, preparation of a remaining-work breakdown, and client approval.

Table 5: Re-estimation timing distribution across effort-consumption levels

Re-estimation timing	**Consumption threshold**	**Number**	**Percentage**
Very early	($< 50\%$)	321	19.8%
Early	(50–75%)	138	8.5%
Near limit	(75–100%)	404	24.9%
Late	(100–150%)	599	37.0%
Very late	($> 150\%$)	158	9.8%

Discussion. Re-estimation is a common procedure in this study environment, which is positive. However, the main issue is that formal estimate updates are often recorded only after the active estimate has already been consumed. This does not necessarily mean that teams became aware of the additional effort needed too late, as the recorded update may represent the finalization and registration of a broader process involving internal review, preparation of client communication, and client approval. Nevertheless, from a monitoring perspective, the system record still formalizes the revised estimate late, which limits the data usefulness for predictive modeling or active planning and budget-control mechanism.

The late timing explanation could be connected back to the anchoring and optimism effects discussed in Section 2.2. Once an estimate is approved and communicated to a client, it becomes a reference point. Teams may remain anchored to the approved estimate and assume that the remaining work can still be completed within the available budget, even when the current effort-consumption pattern suggests otherwise [16, 13]. This finding is a well-documented failure in behavior across estimation practice [17]. However, this is exactly where data-driven monitoring could prove to be valuable. It could provide teams with an additional warning signal to help them step back from the approved estimate, review the remaining work, and make a more informed decision about whether the estimate is still realistic.

The 24.9% of documented estimate updates that occur between 75% and 100% consumption show that timely estimate updates do already occur to some extent in current practice. However, this is not yet consistent enough, since a large share of updates are still recorded only after the previous active estimate has been exceeded. A 75% consumption signal could therefore be useful as a standardized systematic review point. It would give teams a clear moment to review and assess whether the remaining work still fits within the approved estimate, prepare client communication if needed, and document the start of the re-estimation review process before the approved budget is consumed.

6.2 Early-Warning Model Analysis

6.2.1 RQ4: Cross-validation Performance Comparison

Results. We compare three Logistic Regression models with different feature sets, represented in Table 3, with Table 6 showing the grouped cross-validation performance across the five folds. The baseline model achieves the strongest overall performance, with a ROC-AUC of 0.761 ± 0.038, recall of 0.678 ± 0.035, and an F1-score of 0.631 ± 0.048 indicating moderate performance. The extended models performance is similar, given that the standard deviations overlap, despite including additional version-history features. In contrast, the reduced model performs worse on most evaluation metrics, with recall decreasing to 0.346 ± 0.036, the F1-score decreasing to 0.443 ± 0.061, and ROC-AUC decreasing to 0.651 ± 0.046. The only exception is precision, where the reduced model scores slightly higher than the other models. This is most likely because without the consumption ratio the model becomes more cautious and flags fewer cases as at risk overall, which are more likely to be true. However, this comes at the cost of missing many actual overrun-risk cases, as reflected by the lower recall. Given the early-warning context this trade-off is undesirable, as missing a real risk is more problematic than triggering an unnecessary review.

Table 6: Cross-Validation Performance Across Logistic Regression Feature Sets

Metric	**Feature set**		
	Baseline	**Reduced**	**Extended**
ROC-AUC	0.761 ± 0.038	0.651 ± 0.046	0.754 ± 0.037
Accuracy	0.705 ± 0.026	0.676 ± 0.028	0.702 ± 0.026
Precision	0.592 ± 0.069	0.625 ± 0.133	0.591 ± 0.069
Recall	0.678 ± 0.035	0.346 ± 0.036	0.663 ± 0.046
F1-score	0.631 ± 0.048	0.443 ± 0.061	0.623 ± 0.051

The model comparison results suggest that the effort-consumption ratio is central to overrun-risk classification, which is further supported by the coefficient-based feature weight analysis of the baseline model presented in Table 7. In a Logistic Regression model, the coefficients indicate the direction and relative strength of each feature's contribution to the prediction. A positive coefficient means that higher values of the feature increase the predicted likelihood of overrun risk, while a negative coefficient means that higher values decrease the predicted likelihood. The results show that effort-consumption ratio alone accounts for 71.9% of the total absolute coefficient weight, confirming that the baseline model mainly learns overrun risk prediction from a single directly computable relationship of how much of the active estimate has already been consumed.

Table 7: Feature Weight Analysis for the Baseline Logistic Regression Model

Feature	Coefficient	Relative weight
Effort-consumption ratio	1.377	71.9%
Re-estimation count so far	-0.264	13.8%
Current estimate	0.144	7.5%
Re-estimation event indicator	0.070	3.7%
Current time spent	0.059	3.1%

Discussion. This analysis leads to three important findings. The first expected finding is that the version-history data contains a meaningful predictive signal for active-estimate overrun risk. The baseline model achieves a ROC-AUC of 0.761 and a recall of 0.678, which is notable for an exploratory model trained on data from a single project and based only on limited structured effort-consumption features. This suggests that even the basic data that is already recorded during the case life cycle contains useful information for identifying cases that may later exceed their active estimate. Prior work indicates that by further starting to collect, store and incorporate richer contextual features, such as task complexity, developer experience, and code-change metrics, could further strengthen the predictive performance in software estimation and risk detection models [3, 19, 22]. The current absence of such features therefore represents a limitation and a potential direction for future improvement.

The second finding is that adding more structured version-history features does not improve model performance. The extended model performs almost identically to the baseline model, despite including additional variables such as the initial formal estimate, remaining effort, version number, time-spent increase, and whether the case has already been re-estimated. This shows that more features do not automatically lead to better overrun-risk detection or signaling. From a practical perspective, this means that a more complex feature-engineering process would not meaningfully improve detection capability with the structured data currently available.

Lastly, the most important finding is that the effort-consumption ratio alone accounts for 71.9% of what the best-performing baseline model learns. In other words, the model mainly learns overrun risk from a single, directly mathematically computable relationship, which tells us how much effort has already been logged and consumed in relation to the active estimate. This raises an important practical question: if most of the predictive signal comes from the consumption ratio, does the organization need a trained classifier as a first implementation step, or could a simple threshold rule already provide a valuable early-warning value? This motivates and grounds the following threshold analysis and direct performance comparison to the baseline model.

6.2.2 Model Comparison to Consumption-Ratio Warning Thresholds

Results. We evaluate five simple consumption-based threshold rules against the Logistic Regression model, with results presented in Table 8. Each threshold rule flags a case as at risk once its cumulative time spent reaches or exceeds a specified share of the currently active estimate: 50%, 75%, 80%, 85%, or 90%. The threshold comparison clearly shows the expected trade-off between recall and precision. The 50% threshold achieves the highest recall among the thresholds (0.767), because it flags task versions as at risk very early. However, it also has the lowest precision (0.456), meaning that a substantial amount of warnings at this point are false positives. This makes the

50% threshold a too sensitive review trigger to potentially adopt into practice. The 90% threshold shows the opposite pattern. It achieves the highest precision (0.702), because by the time an active estimate has reached 90% consumption, many cases in the dataset have already gone on to exceed their estimate. As a result, warnings generated at this stage are more likely to correspond to genuine overruns, leading to fewer false positives. However, recall drops to 0.484, meaning that many actual overrun-risk observations are missed because they occur before the threshold is reached. This makes the 90% threshold less suitable as an early-warning trigger, as alerts are often generated only when there is little time remaining to review the work, adjust planning, or communicate changes to the client.

The 75% threshold provides the best balance between these two thresholds. At this point, most of the active estimate has already been consumed, so the warning is more meaningful than at 50%. At the same time, the estimate has not yet been fully consumed, so there is still time to review the case, re-estimate the task, or adjust planning before an overrun occurs. The 80% and 85% thresholds further confirm this trade-off: they improve precision and accuracy compared with 75%, but reduce recall and do not improve the F1-score. This makes 75% the most practical simple warning threshold in this comparison. Additionally, compared with the Logistic Regression model, the 75% threshold performs only slightly worse: the Logistic Regression model achieves a precision of 0.592, recall of 0.678, and F1-score of 0.631, while the 75% threshold achieves a precision of 0.556, recall of 0.617, and F1-score of 0.585. Considering this, the performance gap is relatively small, suggesting that the 75% threshold captures much of the same warning signal as the trained classifier.

Table 8: Comparison between Logistic Regression and consumption-ratio warning thresholds

Method	**Precision**	**Recall**	**F1-score**
Logistic Regression	0.592 ± 0.069	0.678 ± 0.035	0.631 ± 0.048
Threshold ≥ 0.50	0.456 ± 0.042	0.767 ± 0.038	0.571 ± 0.034
Threshold ≥ 0.75	0.556 ± 0.052	0.617 ± 0.053	0.584 ± 0.042
Threshold ≥ 0.80	0.595 ± 0.054	0.572 ± 0.056	0.582 ± 0.045
Threshold ≥ 0.85	0.641 ± 0.054	0.531 ± 0.054	0.580 ± 0.047
Threshold ≥ 0.90	0.702 ± 0.045	0.484 ± 0.055	0.572 ± 0.049

Discussion. The Logistic Regression model outperforms all threshold rules, but has only slight improvements compared to the 75% threshold rule performance. This reveals that most of the predictive signal is already contained in the consumption ratio itself, and a simple threshold captures nearly as much of it as a trained classifier.

The most actionable outcome of this analysis is therefore not the model, but instead the directly applicable and practical value of the 75% consumption threshold as an organizational monitoring mechanism. It requires no trained model, retraining process, or complex feature engineering, and relies on only two attributes that are already recorded in the project management system: the active estimate and cumulative time spent. A dashboard flag at this point would suggest cases for human review while the estimate has not yet been exceeded, giving teams time to reassess remaining work, revise the estimate, or communicate with the client before an overrun occurs. This directly addresses the structural timing problem identified in RQ3, where the majority of re-estimations currently occur only after the active estimate has already been consumed. Introducing a systematic

75% review trigger would shift re-estimation from a reactive correction into a proactive planning step. Furthermore, as Jørgensen and Sjøberg note, effort estimates actively shape project behavior rather than just represent expected effort [14]. Therefore, a visible consumption signal could help reframe how remaining budget is perceived by teams.

In practice, the 75% threshold would function as a recurring review trigger rather than a one-time warning. If a re-estimation following a 75% warning adds sufficient hours, the consumption ratio drops back below 75%. The same threshold may then trigger again later if effort consumption continues to rise, creating a repeated review cycle throughout the case life cycle. A potential concern is that this could generate multiple review points for the same case. However, cases that repeatedly approach their active estimate are also the cases carrying higher overrun risk. In such cases, repeated review is not necessarily a drawback, but a signal that the estimate may need to be reassessed more carefully.

Both approaches share a common limitation. The threshold rule and the Logistic Regression model deal with version-level effort-consumption snapshots without any contextual knowledge of where a case is in its actual development progress. A case at 75% consumption that is nearly complete does not carry the same risk as a case at 75% consumption that is only one week into a multi-month effort, although the current model would flag them identically. Catana and Florescu [3] address a comparable limitation in their framework by incorporating task complexity and team experience alongside consumption signals, suggesting that contextual features could help distinguish genuinely at-risk snapshots from cases that are simply large but progressing normally. Without such features, both the threshold rule and the model should be treated as blind review triggers rather than automated re-estimation decisions, requiring a human assessment step before any planning or client communication action is taken.

A further concern is data quality. Both approaches require the active estimate and cumulative time spent to be recorded consistently and progressively during execution. The data exploration in Section 6.1.1 already shows that a large share of cases in the raw export lack one or both of these attributes, limiting the analysis to cases where the required information is available. If time registration is delayed or corrected only near case closure, the consumption ratio at any given snapshot may not reflect the true state of the task at that moment, reducing the reliability of any real-time warning signal. In addition, because many re-estimations currently occur only after the active estimate has already been exceeded, the dataset reflects a largely reactive process. Before developing a more advanced predictive model, the organization would therefore first need more consistent and timely registration of effort, estimates, and review decisions, as prior research also shows that incomplete or inconsistent effort data reduces the reliability of data-driven approaches [2, 10].

6.3 Practitioner Validation and Interpretation

6.3.1 Effort Estimates, Overruns, and Re-estimation Patterns

Both practitioners confirmed that effort estimates function as client-facing budget authorizations rather than only internal planning guidelines. In the studied context, an approved estimate covers the expected effort for analysis, development, implementation, testing, and delivery of a task. Exceeding the active estimate therefore creates a budget-control issue, where additional effort requires renewed client approval before it can be formally registered and used. Practitioners also noted that developers generally aim to complete work within the original estimate, meaning that

re-estimation is treated as a necessary correction rather than as a routine planning step.

When re-estimation does become necessary, the client approval process creates significant operational delay. Responses and confirmations can take weeks, and submitting a request requires preparing an itemized breakdown of remaining work, which practitioners describe as additional administrative burden. During this waiting period, high-priority cases typically continue accumulating effort while approval is still pending, whereas lower-priority cases are completely placed on hold. In both scenarios, the moment of re-estimation recorded in the system reflects the moment of client approval rather than the moment overrun risk was first recognized by the team. This provides important context for the RQ3 finding that 46.8% of re-estimations occur after the active estimate has already been exceeded. Therefore, the late re-estimations also reflect client approval lags.

At the same time, the practitioners identified the fundamental cause underlying both the late re-estimation and the operational issues it causes. Developers currently do not continuously track estimate consumption during active work, and the remaining budget typically only becomes visible at the moment hours are registered in the system. Progress awareness is guided purely by individual professional judgment, where developers draw from their own experience and skill to determine whether a task is on track. While this gut feeling improves with experience, it is completely dependent on individual skill level and cannot be consistently translated into a reliable early-warning signal across a team with varying levels of experience. As a result, the need for re-estimation is often recognized too late to initiate client communication while budget still remains, leaving teams with no choice but to either continue working without approval or stop entirely while waiting for the clients response. However, the practitioners did acknowledge that if consumption were actively tracked and visible during execution, it could create more structured review points, and therefore re-estimation requests could be submitted earlier while effort budget is still available, allowing client communication to happen before work needs to be paused, avoiding the approval-lag problem. This provides partial support for H1, as it suggests that late re-estimation is mainly caused by limited visibility of effort consumption rather than estimation inaccuracy alone, in combination with client delays in response times.

These findings strengthen and motivate the need for consumption-based monitoring. A visible dashboard review trigger would not replace developer judgment, but could support it by creating an earlier and more consistent moment to pause and assess remaining work, prepare for re-estimation when necessary, and communicate with the client before the approved budget is consumed.

6.3.2 Practitioner Views on Consumption-Based Monitoring

When presented with a consumption-based review trigger, both practitioners responded positively and considered it a practical improvement over the current process. The 75% threshold was viewed as a valuable review point because it would make estimate consumption visible while there is still enough budget remaining to reassess the case, prepare a re-estimation request, and initiate client communication if needed. One practitioner suggested that an even earlier 50% threshold could also be useful as an additional check-in point, particularly for teams with less estimation experience or for cases with higher uncertainty. Finally, the 90% threshold was seen as a suitable final confirmation point, where the team should explicitly assess whether the remaining work can still be completed within the approved budget.

These responses support H2, as both practitioners perceived a structured, consumption-ratio-based review trigger as a feasible and valuable improvement over current practice. They highlighted

that earlier detection increases the possible response options: when overrun risk is identified while budget still remains, teams can reconsider implementation, redistribute work, prepare re-estimation, or communicate with the client before work is interrupted. A structured trigger also reduces reliance on individual judgment by introducing a consistent moment to assess remaining work and decide whether further action is needed before the budget is consumed. These findings demonstrate the perceived feasibility and practical plausibility of the proposed trigger according to two practitioners. However, they do not serve as evidence that such a trigger would, in practice, significantly improve re-estimation timing, client communication, or project outcomes, which would require implementation and evaluation in a real-world setting.

6.3.3 Conditions for Adoption

Both practitioners emphasized that successful adoption of consumption-based monitoring would depend on how visibly and conveniently the signal is integrated into the existing workflow. Since the case-study environment already requires administration across multiple tools, an additional warning mechanism should not require developers to manually check a separate dashboard. Instead, the signal should be integrated directly in the workflow of the responsible developer. One practitioner suggested that a notification trigger would be practical when a threshold is crossed, because it would make the risk visible immediately after time registration

The second condition is related to data quality and process consistency. The practitioners noted that effort-registration practices and project-management tools can differ substantially between teams and projects. However, a consumption-based trigger is only useful if active estimates and cumulative time spent are recorded accurately and updated during execution. Therefore, teams need to understand that timely time registration is not only an administrative requirement, but a necessary prerequisite for reliable early-warning signals. Without reliable input data, the triggers may not reflect the actual state of the task.

6.4 Practical Recommendations

Based on the combined findings of the quantitative analysis, threshold comparison, and practitioner interviews, we propose the following recommendations to integrate and support earlier overrun awareness in the case-study environment. They focus on practical process improvements that can be implemented using data already available in the project management system, without requiring a trained model or additional tools.

Process Recommendations

- **Improve baseline data quality before further automation.** Across the raw export, 30.6% Change cases are excluded during preprocessing due to duplication, incomplete histories, or missing/invalid effort estimates and time-spent values, as shown in Table 2. These values should be recorded consistently and completely, as reliable data is required for accurate monitoring, meaningful analysis, and application of the proposed approach.
- **Integrate consumption monitoring visibly directly into the existing case-registration workflow, not as a separate dashboard.** Rather than relying on a separate dashboard, the active consumption ratio should be visible directly within the

case view used by developers during execution. Following each time registration, the ratio should update automatically to indicate the proportion of the active estimate that has been consumed. Such a simple progress indicator could provide continuous awareness of estimate consumption.

- **Adopt a tiered consumption-risk categorization for ongoing cases rather than a single warning point.** Classifying active Change cases into five consumption-risk categories based on the effort-consumption ratio, such as Very Low (¡50%), Early Warning (50–75%), Review (75–90%), Critical (90–100%), and Overrun (¿100%), would enable practitioners to prioritize cases according to urgency. This supports efficient triage by focusing immediate attention on Critical and Overrun cases while providing early visibility of cases approaching higher risk levels. This proposed categorization is adopted in the dashboard prototype presented in Figure 9.

- **Use 75% as the primary review trigger and 90% as the final review trigger within this tiered structure.** When a case reaches 75% of its active estimate, the responsible team member should review the remaining work and assess whether completion within the current estimate is still realistic, and potentially request for additional budget. The 90% trigger should prompt a final assessment point before the active estimate is fully consumed. At this point, the team should confirm that the remaining work can still realistically be completed within the active estimate. If not, formal re-estimation, planning adjustment, or client communication should be initiated immediately before the budget is exceeded.

- **Record the start of the estimate-review process separately from the final estimate update.** This would make it possible to distinguish internal review time from client-approval delay, and to establish a reliable benchmark for how long client confirmation typically takes. Over time, this benchmark could help teams anticipate response delays and decide earlier when a re-estimation request should be submitted.

- **Treat every tier as a review trigger requiring human judgment, not an automated re-estimation decision.** Thresholds should initiate a human intervention and assessment step rather than trigger automatic re-estimation, preserving professional judgment while making overrun risk more visible and systematic.

- **Start with the tiered consumption-ratio rule before adopting a trained model.** Because the consumption ratio captures most of the useful warning signal for overrun risk, a simple threshold-based dashboard would be a practical first implementation step. More complex models should only be considered later if consistent, and richer contextual data, such as task complexity, remaining work, or development progress is recorded and available.

Based on these recommendations, we designed a prototype dashboard in Power BI to demonstrate how the proposed tiered, consumption-based monitoring approach could function in practice. The dashboard, shown in Figure 9, classifies all active Change cases into the five consumption-risk categories described earlier and orders them by descending risk. Overrun and Critical cases are

displayed at the top of the case list, enabling team leads and Scrum Masters to prioritize the most urgent cases, while Review, Early Warning, and Very Low cases remain visible for continued monitoring. The dashboard also allows practitioners to filter the case list by consumption-risk category, to focus on specific groups of cases, such as only Critical and Overrun cases or only those in the Review category. Summary cards at the top of the dashboard provide an overview of the number of active cases within each risk category, allowing practitioners to quickly assess the overall distribution of case risk without inspecting individual cases. Additionally, in a practical setting, notifications to responsible team members regarding task consumption could be added when cases move into higher consumption-risk categories for increased awareness and visibility.

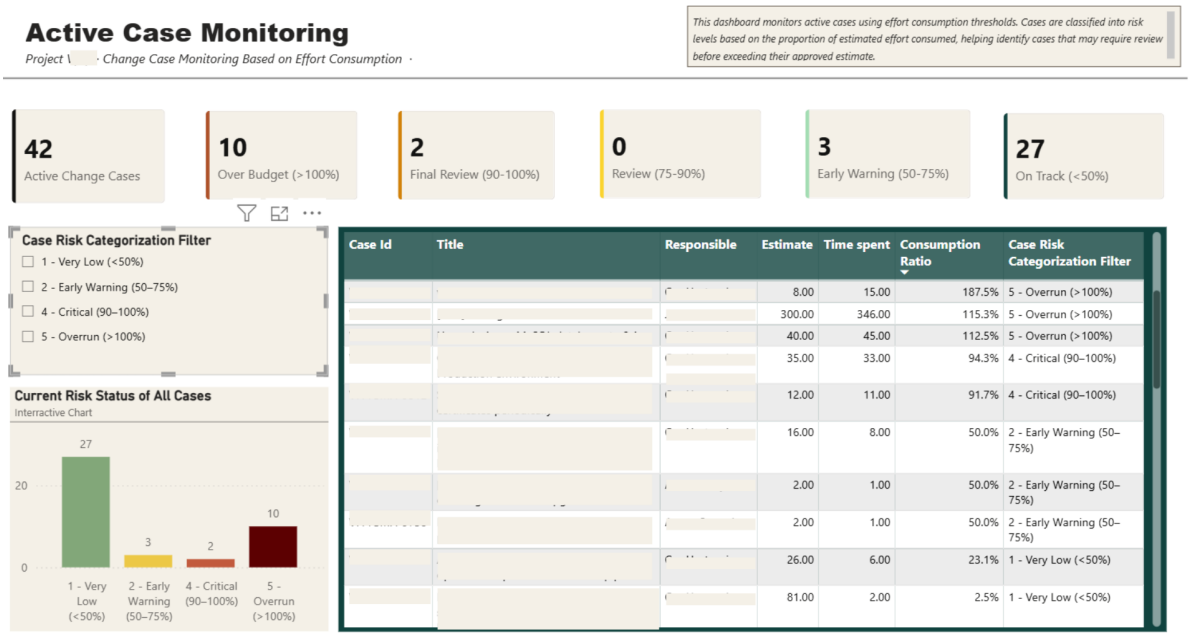


Figure 9: Prototype dashboard showing tiered consumption-based monitoring of Change cases with anonymized data.

6.5 Threats to Validity and Limitations

Several limitations should be considered when interpreting the results of this study. The main threats to validity concern generalizability, the limited empirical grounding of both the quantitative and qualitative components, the gap between administrative and operational reality in the data, and the construct validity of the re-estimation timing analysis.

First, this study is based on a single long-running software maintenance project within one IT consultancy organization. The project operates under a specific contractual structure in which Change cases require client approval for additional effort. The underestimation rate, active-estimate overrun frequency, documented estimate-update behavior, and model performance reported in this study are therefore specific to this context. They may not generalize directly to other maintenance projects, organizations, contract types, or development environments where estimates are used differently. The results should therefore be interpreted as case-specific rather than broadly applicable, and any organization considering the proposed consumption-based triggers should first validate

the 75% and 90% thresholds against their own historical data rather than assuming they transfer directly.

Second, the empirical grounding of both components of this study is limited and narrow. On the quantitative side, after preprocessing, 69.4% of cases remain for analysis. This means that the results may be biased toward better-documented work and may therefore underestimate estimation and tracking issues in practice. In particular, excluding cases with missing or inconsistent estimate and time-spent data may bias the reported overrun statistics, as these cases are likely to represent more poorly tracked or less controlled work, which may also be more prone to estimate overruns. If this is the case, the reported 34.9% active-estimate overrun rate may underestimate, rather than overestimate, the true overrun rate across all Change cases. On the qualitative side, the validation is based on only two interviews from the same project, providing within-case validation rather than independent confirmation. The two practitioners were selected based on availability and direct project experience rather than random or stratified sampling, meaning their views may not represent the full range of opinions across the team, especially regarding willingness to adopt additional monitoring tools. Practitioners against process change, or with different levels of estimation experience, may have responded differently.

Third, the study relies on historical registration data from the project-management system. This means that the results depend on how accurately effort estimates, time spent, estimate updates, and case histories were recorded. Effort estimates are often rounded, time spent may be registered after work has already taken place, and some time-spent values may be corrected retroactively. As a result, the reconstructed effort-consumption ratios may not always reflect the exact state of a case at the moment work was performed. This affects the active-estimate overrun analysis, the documented estimate-update timing analysis, and the performance of the prototype, since any real-world deployment would include the same registration delays. Additionally, model and threshold comparisons in this study rely on cross-validated means and standard deviations rather than formal statistical significance testing. The observed performance differences, such as between the Logistic Regression model and the 75% threshold, should therefore be interpreted descriptively rather than as statistically confirmed differences.

Lastly, an important construct validity threat concerns the interpretation of re-estimation timing. The available data records when an estimate is formally updated in the system, but it does not record when developers first became aware that the active estimate might be insufficient, when an internal estimate review was started, or when a re-estimation request was submitted for client approval. Practitioner input confirmed that formal estimate updates may occur only after internal assessment, preparation of a remaining-work breakdown, and client approval. As a result, RQ3 reflects system update timing rather than true awareness timing, and may make re-estimation appear later than it actually occurs in practice. This distinction matters, as it means that the 46.8% late re-estimation rate should not be read as evidence that teams are unaware of overrun risk, but as evidence that the system does not currently capture awareness until it is formalized into an approved estimate change.

7 Conclusions

In this study we examined how accurately task-level effort estimates reflect actual consumed effort in a client-facing software maintenance project, and whether routinely collected effort-consumption

data can support earlier awareness and identification of tasks at risk of overruns.

The findings show that underestimation is common and that active-estimate overruns occur in roughly one third of completed Change cases, accounting for 14.4% of all logged effort across five years. While re-estimation is practiced regularly, 46.8% of documented estimate updates are recorded only after the active estimate has already been exceeded. Taken individually, these findings could be interpreted as estimation-accuracy problems. Together, however, they indicate to a deeper process issue explanation: overruns are driven less by estimation inaccuracy alone, and more by the absence of a visible, structured checkpoint during task execution. Developers currently rely on professional judgment and experience to assess whether remaining work still fits within the approved budget. Because consumption is not continuously visible, this judgment is applied inconsistently, and formal estimate updates only become visible in the system once the previous active estimate has already been consumed.

The immediate practical implication is therefore not to introduce a complex predictive model for task effort overruns, but to improve consumption visibility and review timing. The quantitative analysis shows that a simple 75% consumption threshold performs close to a trained Logistic Regression classifier while using only data already recorded in the project-management system. Making this visible to developers during active work, and introducing effort consumption ratio mappings and structured review triggers at 75% and 90% consumption, would create the feedback loop that is currently missing. At the 75% point, the team still has approved budget remaining to reassess the task, review the implementation approach, redistribute work, reduce scope, or prepare a re-estimation request and initiate client communication before approval becomes urgent.

The main conclusion of this study is therefore that effort overrun monitoring in software maintenance does not need to begin with a complex predictive model. The signal needed to support earlier awareness is already present in data that most maintenance organizations, including the one studied here, already routinely collect; what is missing is not data, but visibility. A simple, transparent, and workflow-integrated consumption review trigger can already help shift re-estimation from a reactive system update toward a more proactive planning and communication step, without requiring new tooling, new data collection, or specialized modeling expertise to get started.

7.1 Further Work

Future work could first focus on validating the proposed consumption-based review mechanism in a real-world setting. Implementing a lightweight prototype within the project-management workflow would allow evaluation of whether visible consumption ratios and 75%/90% review triggers improve early awareness, re-estimation timing, and client communication in practice. Second, future work should improve the documentation process of re-estimation timings, and activity. The current system only captures when an estimate is formally updated, but not when internal review begins, how long it takes to get approvals or when a re-estimation request is prepared. Introducing separate timestamps for these stages would allow a clearer distinction between internal decision-making time and client approval time, and would enable more precise analysis of re-estimation delays. Finally, future work should assess whether the findings generalize beyond a single project. Applying the same analysis to multiple teams or projects would show whether observed overrun patterns, estimation bias, and the effectiveness of the 75% threshold are context-specific or generalizable and more widely applicable. If richer contextual data (e.g., task complexity or development progress)

becomes available, more advanced predictive models could also be evaluated against the simple consumption-based approach proposed in this study.

References

[1] V. Khatibi Bardsiri, D.N.A. Jawawi, S.Z.M. Hashim, and E. Khatibi. Increasing the accuracy of software development effort estimation using projects clustering. *IET Software*, 6:461–473, 2012.

[2] Michael F. Bosu and Stephen G. Macdonell. Experience: Quality benchmarking of datasets used in software effort estimation. *J. Data and Information Quality*, 11(4), August 2019.

[3] Andreea-Elena Catana and Adriana Florescu. Machine learning-based effort prediction and early risk detection in software development projects: A case study. *International Journal of Advanced Computer Science and Applications*, 17, 01 2026.

[4] Narciso Cerpa, Matthew Bardeen, Barbara Kitchenham, and June Verner. Evaluating logistic regression models to estimate software project outcomes. *Information and Software Technology*, 52(9):934–944, 2010.

[5] Ali Dehghan, Kelly Blincoe, and Daniela Damian. A hybrid model for task completion effort estimation. In *Proceedings of the 2nd International Workshop on Software Analytics*, SWAN 2016, page 22–28, New York, NY, USA, 2016. Association for Computing Machinery.

[6] R. Dillibabu and K. Krishnaiah. Cost estimation of a software product using cocomo ii.2000 model – a case study. *International Journal of Project Management*, 23(4):297–307, 2005.

[7] Srdjana Dragicevic, Stipe Celar, and Mili Turic. Bayesian network model for task effort estimation in agile software development. *Journal of Systems and Software*, 127:109–119, 2017.

[8] Marta Fernández-Diego, Erwin R. Méndez, Fernando González-Ladrón-De-Guevara, Silvia Abrahão, and Emilio Insfran. An update on effort estimation in agile software development: A systematic literature review. *IEEE Access*, 8:166768–166800, 2020.

[9] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software development*, 9(8):28–35, 2001.

[10] Mohd Haleem, Mohammad Islam, Mohammad Nadeem Ahmed, Nafees Akhter Farooqui, Ahmad Neyaz Khan, Mohammad Rashid Hussain, J. Shanawaz Ahmed, Mohammed Mohsin Ahmed, and Imran Mohd Khan. A critical review for software maintenance cost estimation models: A data driven approach. *IEEE Access*, 13:184825–184849, 2025.

[11] M. Jorgensen and K. Molokken-Ostvold. Reasons for software effort estimation error: impact of respondent role, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering*, 30(12):993–1007, 2004.

[12] Magne Jørgensen. A critique of how we measure and interpret the accuracy of software development effort estimation, 2007.

[13] Magne Jørgensen. Identification of more risks can lead to increased over-optimism of and over-confidence in software development effort estimates. *Information and Software Technology*, 52(5):506–516, 2010. TAIC-PART 2008.

[14] Magne Jørgensen and Dag I.K Sjøberg. Impact of effort estimates on software project work. *Information and Software Technology*, 43(15):939–948, 2001.

[15] Sheo Kumar, Sugandha Chakraverti, S Agarwal, and Ashish Chakraverti. Modified cocomo model for maintenance cost estimation of real time system software. *International Journal of Computer Science and Network*, 1, 02 2012.

[16] Erik Løhre and Magne Jørgensen. Numerical anchors and their strong effects on software development effort estimates. *Journal of Systems and Software*, 116:49–56, 2016.

[17] Rahul Mohanani, Iflaah Salman, Burak Turhan, Pilar Rodríguez, and Paul Ralph. Cognitive biases in software engineering: A systematic mapping study. *IEEE Transactions on Software Engineering*, 46(12):1318–1339, 2020.

[18] Ofer Morgenshtern, Tzvi Raz, and Dov Dvir. Factors affecting duration and effort estimation errors in software development projects. *Information and Software Technology*, 49(8):827–837, 2007.

[19] Bruno Budel Rossi and Lisandra Manzoni Fontoura. Ai-based approaches for software tasks effort estimation: A systematic review of methods and trends. *ICEIS (2)*, pages 144–151, 2025.

[20] J. Edward Russo. Understanding the effect of a numerical anchor. *Journal of Consumer Psychology*, 20(1):25–27, 2010.

[21] H.M. Sneed. A cost model for software maintenance & evolution. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 264–273, 2004.

[22] André O. Sousa, Daniel T. Veloso, Henrique M. Gonçalves, João Pascoal Faria, João Mendes-Moreira, Ricardo Graça, Duarte Gomes, Rui Nuno Castro, and Pedro Castro Henriques. Applying machine learning to estimate the effort and duration of individual tasks in software projects. *IEEE Access*, 11:89933–89946, 2023.

[23] Muhammad Usman, Emilia Mendes, and Jürgen Börstler. Effort estimation in agile software development: a survey on the state of the practice. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, New York, NY, USA, 2015. Association for Computing Machinery.

[24] Muhammad Usman, Kai Petersen, Jürgen Börstler, and Pedro Santos Neto. Developing and using checklists to improve software effort estimation: A multi-case study. *Journal of Systems and Software*, 146:286–309, 2018.

[25] Duane Wegener, Richard Petty, Kevin Blankenship, and Brian Detweiler-Bedell. Elaboration and numerical anchoring: Breadth, depth, and the role of (non-)thoughtful processes in anchoring theories. *Journal of Consumer Psychology - J CONSUM PSYCHOL*, 20:28–32, 01 2010.